

RTI Connext C API

Version 7.6.0

1 RTI Connex	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 Data Structure Index	9
3.1 Data Structures	9
4 File Index	23
4.1 File List	23
5 Module Documentation	25
5.1 Clock Selection	25
5.1.1 Available Clocks	25
5.1.2 Clock Selection Strategy	26
5.1.3 Configuring Clock Selection	26
5.2 Domain Module	26
5.2.1 Detailed Description	27
5.3 DomainParticipantFactory	27
5.3.1 Detailed Description	31
5.3.2 Macro Definition Documentation	31
5.3.2.1 DDS_DomainParticipantFactoryQos_INITIALIZER	31
5.3.2.2 DDS_TheParticipantFactory	32
5.3.3 Typedef Documentation	32
5.3.3.1 DDS_DomainParticipantFactory	32
5.3.3.2 DDS_DomainParticipantFactory_RegisterTypeFunction	33
5.3.4 Function Documentation	33
5.3.4.1 DDS_DomainParticipantFactoryQos_equals()	33
5.3.4.2 DDS_DomainParticipantFactoryQos_print()	34
5.3.4.3 DDS_DomainParticipantFactoryQos_to_string()	34
5.3.4.4 DDS_DomainParticipantFactoryQos_to_string_w_params()	35
5.3.4.5 DDS_DomainParticipantFactoryQos_initialize()	36
5.3.4.6 DDS_DomainParticipantFactoryQos_finalize()	36
5.3.4.7 DDS_DomainParticipantFactoryQos_copy()	37
5.3.4.8 DDS_DomainParticipantFactory_get_instance()	38
5.3.4.9 DDS_DomainParticipantFactory_finalize_instance()	38
5.3.4.10 DDS_DomainParticipantFactory_set_default_participant_qos()	39

5.3.4.11 DDS_DomainParticipantFactory_set_default_participant_qos_with_profile()	40
5.3.4.12 DDS_DomainParticipantFactory_get_default_participant_qos()	41
5.3.4.13 DDS_DomainParticipantFactory_create_participant()	41
5.3.4.14 DDS_DomainParticipantFactory_create_participant_with_profile()	43
5.3.4.15 DDS_DomainParticipantFactory_delete_participant()	44
5.3.4.16 DDS_DomainParticipantFactory_lookup_participant()	45
5.3.4.17 DDS_DomainParticipantFactory_get_qos()	45
5.3.4.18 DDS_DomainParticipantFactory_set_qos()	46
5.3.4.19 DDS_DomainParticipantFactory_load_profiles()	46
5.3.4.20 DDS_DomainParticipantFactory_reload_profiles()	47
5.3.4.21 DDS_DomainParticipantFactory_unload_profiles()	48
5.3.4.22 DDS_DomainParticipantFactory_set_default_library()	48
5.3.4.23 DDS_DomainParticipantFactory_set_default_profile()	49
5.3.4.24 DDS_DomainParticipantFactory_get_default_library()	50
5.3.4.25 DDS_DomainParticipantFactory_get_default_profile()	50
5.3.4.26 DDS_DomainParticipantFactory_get_default_profile_library()	51
5.3.4.27 DDS_DomainParticipantFactory_get_participant_factory_qos_from_profile()	51
5.3.4.28 DDS_DomainParticipantFactory_get_participant_qos_from_profile()	52
5.3.4.29 DDS_DomainParticipantFactory_get_publisher_qos_from_profile()	53
5.3.4.30 DDS_DomainParticipantFactory_get_subscriber_qos_from_profile()	53
5.3.4.31 DDS_DomainParticipantFactory_get_datareader_qos_from_profile()	54
5.3.4.32 DDS_DomainParticipantFactory_get_datareader_qos_from_profile_w_topic_name()	54
5.3.4.33 DDS_DomainParticipantFactory_get_datawriter_qos_from_profile()	55
5.3.4.34 DDS_DomainParticipantFactory_get_datawriter_qos_from_profile_w_topic_name()	56
5.3.4.35 DDS_DomainParticipantFactory_get_topic_qos_from_profile()	56
5.3.4.36 DDS_DomainParticipantFactory_get_topic_qos_from_profile_w_topic_name()	57
5.3.4.37 DDS_DomainParticipantFactory_get_qos_profile_libraries()	57
5.3.4.38 DDS_DomainParticipantFactory_get_qos_profiles()	58
5.3.4.39 DDS_DomainParticipantFactory_unregister_thread()	58
5.3.4.40 DDS_DomainParticipantFactory_get_typecode_from_config()	59
5.3.4.41 DDS_DomainParticipantFactory_create_participant_from_config()	59
5.3.4.42 DDS_DomainParticipantFactory_create_participant_from_config_w_params()	61
5.3.4.43 DDS_DomainParticipantFactory_lookup_participant_by_name()	61
5.3.4.44 DDS_DomainParticipantFactory_register_type_support()	62
5.3.4.45 DDS_DomainParticipantFactory_get_participants()	63
5.3.4.46 DDS_DomainParticipantFactory_set_thread_factory()	64
5.3.5 Variable Documentation	65
5.3.5.1 DDS_PARTICIPANT_QOS_DEFAULT	65
5.3.5.2 DDS_PARTICIPANT_CONFIG_PARAMS_DEFAULT	66

5.4 DomainParticipants	66
5.4.1 Detailed Description	75
5.4.2 Macro Definition Documentation	75
5.4.2.1 DDS_INVALID_LOCAL_ADVANCE_NOTICE_STATUS_EXPIRATION_MESSAGE_INITIALIZER	75
5.4.2.2 DDS_DomainParticipantListener_INITIALIZER	75
5.4.2.3 DDS_DomainParticipantQos_INITIALIZER	76
5.4.2.4 DDS_DomainParticipantProtocolStatus_INITIALIZER	76
5.4.3 Typedef Documentation	77
5.4.3.1 DDS_DomainId_t	77
5.4.3.2 DDS_DomainParticipantListener_InvalidLocalAdvanceNoticeStatusCallback	77
5.4.3.3 DDS_DomainParticipant	77
5.4.4 Function Documentation	79
5.4.4.1 DDS_InvalidLocalAdvanceNoticeStatus_equals()	79
5.4.4.2 DDS_DomainParticipantQos_equals()	79
5.4.4.3 DDS_DomainParticipantQos_print()	80
5.4.4.4 DDS_DomainParticipantQos_to_string()	80
5.4.4.5 DDS_DomainParticipantQos_to_string_w_params()	81
5.4.4.6 DDS_DomainParticipantQos_initialize()	82
5.4.4.7 DDS_DomainParticipantQos_finalize()	82
5.4.4.8 DDS_DomainParticipantQos_copy()	83
5.4.4.9 DDS_DomainParticipant_as_entity()	84
5.4.4.10 DDS_DomainParticipantProtocolStatus_initialize()	84
5.4.4.11 DDS_DomainParticipantProtocolStatus_copy()	85
5.4.4.12 DDS_DomainParticipantProtocolStatus_finalize()	85
5.4.4.13 DDS_DomainParticipantProtocolStatus_equals()	86
5.4.4.14 DDS_DomainParticipant_get_default_topic_qos()	86
5.4.4.15 DDS_DomainParticipant_set_default_topic_qos()	87
5.4.4.16 DDS_DomainParticipant_set_default_topic_qos_with_profile()	88
5.4.4.17 DDS_DomainParticipant_get_default_publisher_qos()	89
5.4.4.18 DDS_DomainParticipant_set_default_publisher_qos()	90
5.4.4.19 DDS_DomainParticipant_set_default_publisher_qos_with_profile()	90
5.4.4.20 DDS_DomainParticipant_get_default_datawriter_qos()	92
5.4.4.21 DDS_DomainParticipant_set_default_datawriter_qos()	92
5.4.4.22 DDS_DomainParticipant_set_default_datawriter_qos_with_profile()	93
5.4.4.23 DDS_DomainParticipant_get_default_subscriber_qos()	94
5.4.4.24 DDS_DomainParticipant_set_default_subscriber_qos()	95
5.4.4.25 DDS_DomainParticipant_set_default_subscriber_qos_with_profile()	95
5.4.4.26 DDS_DomainParticipant_get_default_datareader_qos()	96
5.4.4.27 DDS_DomainParticipant_set_default_datareader_qos()	97

5.4.4.28 DDS_DomainParticipant_set_default_datareader_qos_with_profile()	98
5.4.4.29 DDS_DomainParticipant_get_default_flowcontroller_property()	98
5.4.4.30 DDS_DomainParticipant_set_default_flowcontroller_property()	99
5.4.4.31 DDS_DomainParticipant_get_default_library()	100
5.4.4.32 DDS_DomainParticipant_get_default_profile()	100
5.4.4.33 DDS_DomainParticipant_get_default_profile_library()	101
5.4.4.34 DDS_DomainParticipant_set_default_library()	101
5.4.4.35 DDS_DomainParticipant_set_default_profile()	102
5.4.4.36 DDS_DomainParticipant_create_publisher()	103
5.4.4.37 DDS_DomainParticipant_create_publisher_with_profile()	104
5.4.4.38 DDS_DomainParticipant_delete_publisher()	105
5.4.4.39 DDS_DomainParticipant_create_subscriber()	106
5.4.4.40 DDS_DomainParticipant_create_subscriber_with_profile()	107
5.4.4.41 DDS_DomainParticipant_delete_subscriber()	108
5.4.4.42 DDS_DomainParticipant_create_datawriter()	109
5.4.4.43 DDS_DomainParticipant_create_datawriter_with_profile()	110
5.4.4.44 DDS_DomainParticipant_delete_datawriter()	111
5.4.4.45 DDS_DomainParticipant_create_datareader()	112
5.4.4.46 DDS_DomainParticipant_create_datareader_with_profile()	113
5.4.4.47 DDS_DomainParticipant_delete_datareader()	114
5.4.4.48 DDS_DomainParticipant_create_topic()	115
5.4.4.49 DDS_DomainParticipant_create_topic_with_profile()	116
5.4.4.50 DDS_DomainParticipant_delete_topic()	117
5.4.4.51 DDS_DomainParticipant_create_contentfilteredtopic()	118
5.4.4.52 DDS_DomainParticipant_create_contentfilteredtopic_with_filter()	119
5.4.4.53 DDS_DomainParticipant_delete_contentfilteredtopic()	119
5.4.4.54 DDS_DomainParticipant_register_contentfilter()	120
5.4.4.55 DDS_DomainParticipant_lookup_contentfilter()	121
5.4.4.56 DDS_DomainParticipant_unregister_contentfilter()	122
5.4.4.57 DDS_DomainParticipant_create_multitopic()	122
5.4.4.58 DDS_DomainParticipant_delete_multitopic()	123
5.4.4.59 DDS_DomainParticipant_create_flowcontroller()	124
5.4.4.60 DDS_DomainParticipant_delete_flowcontroller()	125
5.4.4.61 DDS_DomainParticipant_get_typecode()	125
5.4.4.62 DDS_DomainParticipant_find_topic()	126
5.4.4.63 DDS_DomainParticipant_lookup_topicdescription()	127
5.4.4.64 DDS_DomainParticipant_lookup_flowcontroller()	128
5.4.4.65 DDS_DomainParticipant_get_participant_protocol_status()	128
5.4.4.66 DDS_DomainParticipant_get_builtin_subscriber()	129

5.4.4.67 DDS_DomainParticipant_get_implicit_publisher()	130
5.4.4.68 DDS_DomainParticipant_get_implicit_subscriber()	130
5.4.4.69 DDS_DomainParticipant_ignore_participant()	131
5.4.4.70 DDS_DomainParticipant_ignore_topic()	132
5.4.4.71 DDS_DomainParticipant_ignore_publication()	133
5.4.4.72 DDS_DomainParticipant_ignore_subscription()	133
5.4.4.73 DDS_DomainParticipant_banish_ignored_participants()	134
5.4.4.74 DDS_DomainParticipant_get_domain_id()	135
5.4.4.75 DDS_DomainParticipant_delete_contained_entities()	136
5.4.4.76 DDS_DomainParticipant_assert_liveliness()	136
5.4.4.77 DDS_DomainParticipant_get_publishers()	137
5.4.4.78 DDS_DomainParticipant_get_subscribers()	138
5.4.4.79 DDS_DomainParticipant_get_current_time()	138
5.4.4.80 DDS_DomainParticipant_contains_entity()	139
5.4.4.81 DDS_DomainParticipant_register_durable_subscription()	139
5.4.4.82 DDS_DomainParticipant_delete_durable_subscription()	141
5.4.4.83 DDS_DomainParticipant_resume_endpoint_discovery()	141
5.4.4.84 DDS_DomainParticipant_set_dns_tracker_polling_period()	143
5.4.4.85 DDS_DomainParticipant_get_dns_tracker_polling_period()	143
5.4.4.86 DDS_DomainParticipant_get_discovered_participants()	144
5.4.4.87 DDS_DomainParticipant_get_discovered_participants_from_subject_name()	144
5.4.4.88 DDS_DomainParticipant_get_discovered_participant_data()	145
5.4.4.89 DDS_DomainParticipant_get_discovered_participant_subject_name()	146
5.4.4.90 DDS_DomainParticipant_get_discovered_topics()	147
5.4.4.91 DDS_DomainParticipant_get_discovered_topic_data()	147
5.4.4.92 DDS_DomainParticipant_take_discovery_snapshot()	148
5.4.4.93 DDS_DomainParticipant_add_peer()	149
5.4.4.94 DDS_DomainParticipant_remove_peer()	150
5.4.4.95 DDS_DomainParticipant_set_qos()	151
5.4.4.96 DDS_DomainParticipant_set_qos_with_profile()	152
5.4.4.97 DDS_DomainParticipant_get_qos()	152
5.4.4.98 DDS_DomainParticipant_set_property()	153
5.4.4.99 DDS_DomainParticipant_set_listener()	154
5.4.4.100 DDS_DomainParticipant_get_listener()	155
5.4.4.101 DDS_DomainParticipant_get_listenerX()	156
5.4.4.102 DDS_DomainParticipant_lookup_publisher_by_name()	156
5.4.4.103 DDS_DomainParticipant_lookup_subscriber_by_name()	157
5.4.4.104 DDS_DomainParticipant_lookup_datawriter_by_name()	158
5.4.4.105 DDS_DomainParticipant_lookup_datareader_by_name()	159

5.4.5 Variable Documentation	160
5.4.5.1 DDS_DOMAINPARTICIPANTFACTORY_QOS_PRINT_ALL	160
5.4.5.2 DDS_DOMAINPARTICIPANT_QOS_PRINT_ALL	160
5.4.5.3 DDS_TOPIC_QOS_DEFAULT	161
5.4.5.4 DDS_PUBLISHER_QOS_DEFAULT	161
5.4.5.5 DDS_SUBSCRIBER_QOS_DEFAULT	162
5.4.5.6 DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT	163
5.4.5.7 DDS_PUBLISHER_QOS_PRINT_ALL	163
5.4.5.8 DDS_SQLFILTER_NAME	164
5.4.5.9 DDS_STRINGMATCHFILTER_NAME	164
5.4.5.10 DDS_SUBSCRIBER_QOS_PRINT_ALL	164
5.4.5.11 DDS_TOPIC_QOS_PRINT_ALL	165
5.5 Built-in Topics	165
5.5.1 Detailed Description	166
5.6 Topic Module	167
5.6.1 Detailed Description	168
5.7 Topics	168
5.7.1 Detailed Description	172
5.7.2 Macro Definition Documentation	172
5.7.2.1 DDS_PrintFormatProperty_INITIALIZER	173
5.7.2.2 DDS_InconsistentTopicStatus_INITIALIZER	173
5.7.2.3 DDS_TopicQos_INITIALIZER	174
5.7.2.4 DDS_TopicListener_INITIALIZER	174
5.7.2.5 DDS_ContentFilter_INITIALIZER	175
5.7.3 Typedef Documentation	175
5.7.3.1 DDS_PrintFormatProperty	175
5.7.3.2 DDS_TopicDescription	175
5.7.3.3 DDS_Topic	176
5.7.3.4 DDS_TopicListener_InconsistentTopicCallback	176
5.7.3.5 DDS_ContentFilteredTopic	177
5.7.3.6 DDS_ContentFilterCompileFunction	177
5.7.3.7 DDS_ContentFilterEvaluateFunction	178
5.7.3.8 DDS_ContentFilterFinalizeFunction	179
5.7.3.9 DDS_ContentFilterWriterFinalizeFunction	180
5.7.3.10 DDS_ContentFilterWriterAttachFunction	180
5.7.3.11 DDS_ContentFilterWriterDetachFunction	181
5.7.3.12 DDS_ContentFilterWriterCompileFunction	182
5.7.3.13 DDS_ContentFilterWriterEvaluateFunction	183
5.7.3.14 DDS_ContentFilterWriterReturnLoanFunction	183

5.7.3.15 DDS_MultiTopic	185
5.7.4 Enumeration Type Documentation	186
5.7.4.1 DDS_PrintFormatKind	186
5.7.5 Function Documentation	186
5.7.5.1 DDS_InconsistentTopicStatus_initialize()	186
5.7.5.2 DDS_InconsistentTopicStatus_copy()	187
5.7.5.3 DDS_InconsistentTopicStatus_finalize()	188
5.7.5.4 DDS_InconsistentTopicStatus_equals()	188
5.7.5.5 DDS_TopicQos_equals()	189
5.7.5.6 DDS_TopicQos_print()	189
5.7.5.7 DDS_TopicQos_to_string()	190
5.7.5.8 DDS_TopicQos_to_string_w_params()	190
5.7.5.9 DDS_TopicQos_initialize()	191
5.7.5.10 DDS_TopicQos_finalize()	192
5.7.5.11 DDS_TopicQos_copy()	193
5.7.5.12 DDS_TopicDescription_get_type_name()	193
5.7.5.13 DDS_TopicDescription_get_name()	194
5.7.5.14 DDS_TopicDescription_get_participant()	195
5.7.5.15 DDS_Topic_as_entity()	195
5.7.5.16 DDS_Topic_as_topicdescription()	196
5.7.5.17 DDS_Topic_narrow()	196
5.7.5.18 DDS_Topic_narrow_from_entity()	196
5.7.5.19 DDS_Topic_get_inconsistent_topic_status()	197
5.7.5.20 DDS_Topic_set_qos()	197
5.7.5.21 DDS_Topic_set_qos_with_profile()	198
5.7.5.22 DDS_Topic_get_qos()	199
5.7.5.23 DDS_Topic_set_listener()	199
5.7.5.24 DDS_Topic_get_listener()	200
5.7.5.25 DDS_Topic_get_listenerX()	200
5.7.5.26 DDS_ContentFilteredTopic_as_topicdescription()	201
5.7.5.27 DDS_ContentFilteredTopic_narrow()	201
5.7.5.28 DDS_ContentFilteredTopic_get_filter_expression()	201
5.7.5.29 DDS_ContentFilteredTopic_get_expression_parameters()	202
5.7.5.30 DDS_ContentFilteredTopic_set_expression_parameters()	203
5.7.5.31 DDS_ContentFilteredTopic_set_expression()	203
5.7.5.32 DDS_ContentFilteredTopic_append_to_expression_parameter()	204
5.7.5.33 DDS_ContentFilteredTopic_remove_from_expression_parameter()	204
5.7.5.34 DDS_ContentFilteredTopic_get_related_topic()	206
5.7.5.35 DDS_MultiTopic_as_topicdescription()	206

5.7.5.36 DDS_MultiTopic_narrow()	207
5.7.5.37 DDS_MultiTopic_get_subscription_expression()	207
5.7.5.38 DDS_MultiTopic_get_expression_parameters()	208
5.7.5.39 DDS_MultiTopic_set_expression_parameters()	208
5.7.6 Variable Documentation	209
5.7.6.1 DDS_PRINT_FORMAT_PROPERTY_DEFAULT	209
5.7.6.2 DDS_FILTERSAMPLEINFO_DEFAULT	209
5.8 FlatData Topic-Types	209
5.9 Zero Copy Transfer Over Shared Memory	210
5.10 User Data Type Support	210
5.10.1 Detailed Description	212
5.10.2 Macro Definition Documentation	213
5.10.2.1 DDS_TYPESUPPORT_C	213
5.10.2.2 DDS_DATAWRITER_C	213
5.10.2.3 DDS_DATAREADER_C	214
5.10.3 Typedef Documentation	214
5.10.3.1 DDS_InstanceHandle_t	214
5.10.3.2 DDS_TypeSupport	215
5.10.4 Function Documentation	215
5.10.4.1 FooTypeSupport_create_data()	215
5.10.4.2 FooTypeSupport_create_data_ex()	215
5.10.4.3 FooTypeSupport_create_data_w_params()	216
5.10.4.4 FooTypeSupport_copy_data()	216
5.10.4.5 FooTypeSupport_delete_data()	217
5.10.4.6 FooTypeSupport_delete_data_ex()	217
5.10.4.7 FooTypeSupport_delete_data_w_params()	218
5.10.4.8 FooTypeSupport_initialize_data()	219
5.10.4.9 FooTypeSupport_initialize_data_ex()	219
5.10.4.10 FooTypeSupport_finalize_data()	220
5.10.4.11 FooTypeSupport_finalize_data_ex()	220
5.10.4.12 FooTypeSupport_get_type_name()	221
5.10.4.13 FooTypeSupport_register_type()	221
5.10.4.14 FooTypeSupport_unregister_type()	222
5.10.4.15 FooTypeSupport_print_data()	223
5.10.4.16 FooTypeSupport_serialize_data_to_cdr_buffer()	223
5.10.4.17 FooTypeSupport_serialize_data_to_cdr_buffer_ex()	224
5.10.4.18 FooTypeSupport_deserialize_data_from_cdr_buffer()	225
5.10.4.19 FooTypeSupport_data_to_string()	225
5.10.4.20 FooTypeSupport_get_typecode()	226

5.10.4.21 DDS_InstanceHandle_equals()	226
5.10.4.22 DDS_InstanceHandle_compare()	227
5.10.4.23 DDS_InstanceHandle_copy()	227
5.10.4.24 DDS_InstanceHandle_is_nil()	228
5.10.5 Variable Documentation	228
5.10.5.1 DDS_HANDLE_NIL	228
5.11 Type Code Support	228
5.11.1 Detailed Description	236
5.11.2 Accessing a Local ::DDS_TypeCode	237
5.11.3 Accessing a Remote ::DDS_TypeCode	237
5.11.4 Macro Definition Documentation	237
5.11.4.1 DDS_TYPECODE_MEMBER_ID_INVALID	238
5.11.4.2 DDS_TYPECODE_INDEX_INVALID	238
5.11.4.3 DDS_TYPECODE_NOT_BITFIELD	238
5.11.4.4 DDS_VM_NONE	238
5.11.4.5 DDS_VM_CUSTOM	239
5.11.4.6 DDS_VM_ABSTRACT	239
5.11.4.7 DDS_VM_TRUNCATABLE	239
5.11.4.8 DDS_PRIVATE_MEMBER	240
5.11.4.9 DDS_PUBLIC_MEMBER	240
5.11.4.10 DDS_TYPECODE_NONKEY_MEMBER	240
5.11.4.11 DDS_TYPECODE_KEY_MEMBER	241
5.11.4.12 DDS_TYPECODE_NONKEY_REQUIRED_MEMBER	241
5.11.4.13 DDS_TypeCode_PrintFormat_INITIALIZER	242
5.11.5 Typedef Documentation	242
5.11.5.1 DDS_ValueModifier	242
5.11.5.2 DDS_Visibility	242
5.11.6 Enumeration Type Documentation	242
5.11.6.1 DDS_TypeCodePrintFormatKind	242
5.11.6.2 DDS_TCKind	243
5.11.6.3 DDS_ExtensibilityKind	244
5.11.7 Function Documentation	244
5.11.7.1 DDS_TypeCode_kind()	244
5.11.7.2 DDS_TypeCode_extensibility_kind()	245
5.11.7.3 DDS_TypeCode_equal()	248
5.11.7.4 DDS_TypeCode_name()	249
5.11.7.5 DDS_TypeCode_default_annotation()	249
5.11.7.6 DDS_TypeCode_member_default_annotation()	250
5.11.7.7 DDS_TypeCode_min_annotation()	250

5.11.7.8 DDS_TypeCode_max_annotation()	250
5.11.7.9 DDS_TypeCode_default_value()	250
5.11.7.10 DDS_TypeCode_min_value()	251
5.11.7.11 DDS_TypeCode_max_value()	252
5.11.7.12 DDS_TypeCode_member_default_value()	252
5.11.7.13 DDS_TypeCode_member_min_value()	253
5.11.7.14 DDS_TypeCode_member_max_value()	254
5.11.7.15 DDS_TypeCode_member_count()	255
5.11.7.16 DDS_TypeCode_member_name()	256
5.11.7.17 DDS_TypeCode_find_member_by_name()	257
5.11.7.18 DDS_TypeCode_member_type()	257
5.11.7.19 DDS_TypeCode_member_label_count()	258
5.11.7.20 DDS_TypeCode_member_label()	259
5.11.7.21 DDS_TypeCode_member_ordinal()	260
5.11.7.22 DDS_TypeCode_is_member_key()	261
5.11.7.23 DDS_TypeCode_is_member_required()	262
5.11.7.24 DDS_TypeCode_is_member_pointer()	263
5.11.7.25 DDS_TypeCode_is_member_bitfield()	263
5.11.7.26 DDS_TypeCode_member_bitfield_bits()	264
5.11.7.27 DDS_TypeCode_member_visibility()	265
5.11.7.28 DDS_TypeCode_discriminator_type()	266
5.11.7.29 DDS_TypeCode_length()	267
5.11.7.30 DDS_TypeCode_array_dimension_count()	268
5.11.7.31 DDS_TypeCode_array_dimension()	268
5.11.7.32 DDS_TypeCode_element_count()	269
5.11.7.33 DDS_TypeCode_content_type()	270
5.11.7.34 DDS_TypeCode_is_alias_pointer()	271
5.11.7.35 DDS_TypeCode_default_index()	271
5.11.7.36 DDS_TypeCode_concrete_base_type()	272
5.11.7.37 DDS_TypeCode_type_modifier()	274
5.11.7.38 DDS_TypeCode_find_member_by_id()	275
5.11.7.39 DDS_TypeCode_member_id()	276
5.11.7.40 DDS_TypeCode_get_type_object_serialized_size()	277
5.11.7.41 DDS_TypeCode_add_member_to_enum()	278
5.11.7.42 DDS_TypeCode_add_member_to_union()	279
5.11.7.43 DDS_TypeCode_add_member()	280
5.11.7.44 DDS_TypeCode_add_member_ex()	282
5.11.7.45 DDS_TypeCode_print_IDL()	284
5.11.7.46 DDS_TypeCode_print()	284

5.11.7.47 DDS_TypeCode_to_string()	285
5.11.7.48 DDS_TypeCode_to_string_w_format()	286
5.11.7.49 DDS_TypeCode_get_cdr_serialized_sample_max_size()	286
5.11.7.50 DDS_TypeCode_cdr_serialized_sample_max_size()	287
5.11.7.51 DDS_TypeCode_cdr_serialized_sample_min_size()	288
5.11.7.52 DDS_TypeCode_cdr_serialized_sample_key_max_size()	288
5.11.7.53 DDS_TypeCodeFactory_get_instance()	289
5.11.7.54 DDS_TypeCodeFactory_finalize_instance()	289
5.11.7.55 DDS_TypeCodeFactory_clone_tc()	290
5.11.7.56 DDS_TypeCodeFactory_delete_tc()	290
5.11.7.57 DDS_TypeCodeFactory_get_primitive_tc()	291
5.11.7.58 DDS_TypeCodeFactory_create_struct_tc()	291
5.11.7.59 DDS_TypeCodeFactory_create_struct_tc_ex()	292
5.11.7.60 DDS_TypeCodeFactory_create_value_tc()	293
5.11.7.61 DDS_TypeCodeFactory_create_value_tc_ex()	293
5.11.7.62 DDS_TypeCodeFactory_create_union_tc()	294
5.11.7.63 DDS_TypeCodeFactory_create_union_tc_ex()	295
5.11.7.64 DDS_TypeCodeFactory_create_enum_tc()	296
5.11.7.65 DDS_TypeCodeFactory_create_enum_tc_ex()	297
5.11.7.66 DDS_TypeCodeFactory_create_alias_tc()	297
5.11.7.67 DDS_TypeCodeFactory_create_string_tc()	298
5.11.7.68 DDS_TypeCodeFactory_create_wstring_tc()	299
5.11.7.69 DDS_TypeCodeFactory_create_sequence_tc()	299
5.11.7.70 DDS_TypeCodeFactory_create_array_tc()	300
5.11.8 Variable Documentation	300
5.11.8.1 DDS_g_tc_null	300
5.11.8.2 DDS_g_tc_short	301
5.11.8.3 DDS_g_tc_long	301
5.11.8.4 DDS_g_tc_ushort	301
5.11.8.5 DDS_g_tc_ulong	302
5.11.8.6 DDS_g_tc_float	302
5.11.8.7 DDS_g_tc_double	302
5.11.8.8 DDS_g_tc_boolean	303
5.11.8.9 DDS_g_tc_char	303
5.11.8.10 DDS_g_tc_octet	303
5.11.8.11 DDS_g_tc_int8	304
5.11.8.12 DDS_g_tc_uint8	304
5.11.8.13 DDS_g_tc_longlong	304
5.11.8.14 DDS_g_tc_ulonglong	305

5.11.8.15 DDS_g_tc_longdouble	305
5.11.8.16 DDS_g_tc_wchar	305
5.12 Built-in Types	305
5.12.1 Detailed Description	306
5.12.2 Managing Memory for Builtin Types	306
5.12.3 Typecodes for Builtin Types	307
5.13 Built-in Topic's Trust Types	309
5.13.1 Detailed Description	310
5.13.2 Typedef Documentation	310
5.13.2.1 DDS_ParticipantTrustAttributesMask	310
5.13.2.2 DDS_PluginParticipantTrustAttributesMask	310
5.13.2.3 DDS_ParticipantTrustProtectionInfo	310
5.13.2.4 DDS_EndpointTrustAttributesMask	311
5.13.2.5 DDS_PluginEndpointTrustAttributesMask	311
5.13.2.6 DDS_VendorEndpointTrustAttributesMask	311
5.13.2.7 DDS_EndpointTrustProtectionInfo	311
5.13.2.8 DDS_TrustAlgorithmBit	311
5.13.2.9 DDS_TrustAlgorithmSet	311
5.13.2.10 DDS_TrustAlgorithmRequirements	311
5.13.2.11 DDS_ParticipantTrustSignatureAlgorithmInfo	312
5.13.2.12 DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo	312
5.13.2.13 DDS_ParticipantTrustInterceptorAlgorithmInfo	312
5.13.2.14 DDS_ParticipantTrustAlgorithmInfo	312
5.13.2.15 DDS_EndpointTrustInterceptorAlgorithmInfo	312
5.13.2.16 DDS_EndpointTrustAlgorithmInfo	313
5.14 Dynamic Data	313
5.14.1 Detailed Description	323
5.14.2 Macro Definition Documentation	324
5.14.2.1 DDS_DynamicDataProperty_t_INITIALIZER	324
5.14.2.2 DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED	325
5.14.2.3 DDS_DynamicDataTypeProperty_t_INITIALIZER	325
5.14.3 Typedef Documentation	325
5.14.3.1 DDS_DynamicDataMemberId	325
5.14.3.2 DDS_DynamicDataWriter	326
5.14.3.3 DDS_DynamicDataReader	326
5.14.3.4 DDS_DynamicDataTypeSupport	326
5.14.4 Function Documentation	327
5.14.4.1 DDS_DynamicData_initialize()	327
5.14.4.2 DDS_DynamicData_new()	327

5.14.4.3 DDS_DynamicData_finalize()	328
5.14.4.4 DDS_DynamicData_delete()	329
5.14.4.5 DDS_DynamicData_copy()	329
5.14.4.6 DDS_DynamicData_equal()	330
5.14.4.7 DDS_DynamicData_clear_all_members()	330
5.14.4.8 DDS_DynamicData_clear_optional_member()	330
5.14.4.9 DDS_DynamicData_clear_member()	331
5.14.4.10 DDS_DynamicData_set_cdr_buffer()	332
5.14.4.11 DDS_DynamicData_get_cdr_buffer()	332
5.14.4.12 DDS_DynamicData_is_cdr()	333
5.14.4.13 DDS_DynamicData_set_discriminator()	334
5.14.4.14 DDS_DynamicData_get_discriminator()	335
5.14.4.15 DDS_DynamicData_print()	336
5.14.4.16 DDS_DynamicData_from_cdr_buffer()	337
5.14.4.17 DDS_DynamicData_to_cdr_buffer_ex()	338
5.14.4.18 DDS_DynamicData_to_cdr_buffer()	338
5.14.4.19 DDS_DynamicData_to_string()	339
5.14.4.20 DDS_DynamicData_from_string()	340
5.14.4.21 DDS_DynamicData_get_info()	340
5.14.4.22 DDS_DynamicData_bind_type()	341
5.14.4.23 DDS_DynamicData_unbind_type()	341
5.14.4.24 DDS_DynamicData_bind_complex_member()	342
5.14.4.25 DDS_DynamicData_unbind_complex_member()	344
5.14.4.26 DDS_DynamicData_get_type()	345
5.14.4.27 DDS_DynamicData_get_type_kind()	345
5.14.4.28 DDS_DynamicData_get_member_count()	345
5.14.4.29 DDS_DynamicData_member_exists()	346
5.14.4.30 DDS_DynamicData_member_exists_in_type()	346
5.14.4.31 DDS_DynamicData_get_member_info()	347
5.14.4.32 DDS_DynamicData_get_member_info_by_index()	348
5.14.4.33 DDS_DynamicData_get_member_type()	349
5.14.4.34 DDS_DynamicData_is_member_key()	350
5.14.4.35 DDS_DynamicData_get_long()	351
5.14.4.36 DDS_DynamicData_get_short()	352
5.14.4.37 DDS_DynamicData_get_ulong()	352
5.14.4.38 DDS_DynamicData_get_ushort()	353
5.14.4.39 DDS_DynamicData_get_float()	354
5.14.4.40 DDS_DynamicData_get_double()	355
5.14.4.41 DDS_DynamicData_get_boolean()	356

5.14.4.42 DDS_DynamicData_get_char()	357
5.14.4.43 DDS_DynamicData_get_octet()	357
5.14.4.44 DDS_DynamicData_get_longlong()	358
5.14.4.45 DDS_DynamicData_get_ulonglong()	359
5.14.4.46 DDS_DynamicData_get_longdouble()	360
5.14.4.47 DDS_DynamicData_get_wchar()	361
5.14.4.48 DDS_DynamicData_get_int8()	362
5.14.4.49 DDS_DynamicData_get_uint8()	362
5.14.4.50 DDS_DynamicData_get_string()	363
5.14.4.51 DDS_DynamicData_get_wstring()	364
5.14.4.52 DDS_DynamicData_get_complex_member()	365
5.14.4.53 DDS_DynamicData_get_long_array()	366
5.14.4.54 DDS_DynamicData_get_short_array()	367
5.14.4.55 DDS_DynamicData_get_ulong_array()	368
5.14.4.56 DDS_DynamicData_get_ushort_array()	369
5.14.4.57 DDS_DynamicData_get_float_array()	370
5.14.4.58 DDS_DynamicData_get_double_array()	371
5.14.4.59 DDS_DynamicData_get_boolean_array()	372
5.14.4.60 DDS_DynamicData_get_char_array()	373
5.14.4.61 DDS_DynamicData_get_octet_array()	374
5.14.4.62 DDS_DynamicData_get_longlong_array()	375
5.14.4.63 DDS_DynamicData_get_ulonglong_array()	376
5.14.4.64 DDS_DynamicData_get_longdouble_array()	377
5.14.4.65 DDS_DynamicData_get_wchar_array()	378
5.14.4.66 DDS_DynamicData_get_int8_array()	379
5.14.4.67 DDS_DynamicData_get_uint8_array()	380
5.14.4.68 DDS_DynamicData_get_long_seq()	381
5.14.4.69 DDS_DynamicData_get_short_seq()	382
5.14.4.70 DDS_DynamicData_get_ulong_seq()	383
5.14.4.71 DDS_DynamicData_get_ushort_seq()	384
5.14.4.72 DDS_DynamicData_get_float_seq()	384
5.14.4.73 DDS_DynamicData_get_double_seq()	385
5.14.4.74 DDS_DynamicData_get_boolean_seq()	386
5.14.4.75 DDS_DynamicData_get_char_seq()	387
5.14.4.76 DDS_DynamicData_get_octet_seq()	388
5.14.4.77 DDS_DynamicData_get_longlong_seq()	389
5.14.4.78 DDS_DynamicData_get_ulonglong_seq()	390
5.14.4.79 DDS_DynamicData_get_longdouble_seq()	390
5.14.4.80 DDS_DynamicData_get_wchar_seq()	391

5.14.4.81 DDS_DynamicData_get_int8_seq()	392
5.14.4.82 DDS_DynamicData_get_uint8_seq()	393
5.14.4.83 DDS_DynamicData_set_long()	394
5.14.4.84 DDS_DynamicData_set_short()	395
5.14.4.85 DDS_DynamicData_set_ulong()	395
5.14.4.86 DDS_DynamicData_set_ushort()	396
5.14.4.87 DDS_DynamicData_set_float()	397
5.14.4.88 DDS_DynamicData_set_double()	397
5.14.4.89 DDS_DynamicData_set_boolean()	398
5.14.4.90 DDS_DynamicData_set_char()	399
5.14.4.91 DDS_DynamicData_set_octet()	400
5.14.4.92 DDS_DynamicData_set_longlong()	400
5.14.4.93 DDS_DynamicData_set_ulonglong()	401
5.14.4.94 DDS_DynamicData_set_longdouble()	402
5.14.4.95 DDS_DynamicData_set_wchar()	402
5.14.4.96 DDS_DynamicData_set_int8()	403
5.14.4.97 DDS_DynamicData_set_uint8()	404
5.14.4.98 DDS_DynamicData_set_string()	405
5.14.4.99 DDS_DynamicData_set_wstring()	405
5.14.4.100 DDS_DynamicData_set_complex_member()	406
5.14.4.101 DDS_DynamicData_set_long_array()	408
5.14.4.102 DDS_DynamicData_set_short_array()	409
5.14.4.103 DDS_DynamicData_set_ulong_array()	409
5.14.4.104 DDS_DynamicData_set_ushort_array()	410
5.14.4.105 DDS_DynamicData_set_float_array()	411
5.14.4.106 DDS_DynamicData_set_double_array()	412
5.14.4.107 DDS_DynamicData_set_boolean_array()	413
5.14.4.108 DDS_DynamicData_set_char_array()	414
5.14.4.109 DDS_DynamicData_set_octet_array()	415
5.14.4.110 DDS_DynamicData_set_longlong_array()	416
5.14.4.111 DDS_DynamicData_set_ulonglong_array()	416
5.14.4.112 DDS_DynamicData_set_longdouble_array()	417
5.14.4.113 DDS_DynamicData_set_wchar_array()	418
5.14.4.114 DDS_DynamicData_set_int8_array()	419
5.14.4.115 DDS_DynamicData_set_uint8_array()	420
5.14.4.116 DDS_DynamicData_set_long_seq()	421
5.14.4.117 DDS_DynamicData_set_short_seq()	422
5.14.4.118 DDS_DynamicData_set_ulong_seq()	422
5.14.4.119 DDS_DynamicData_set_ushort_seq()	423

5.14.4.120 DDS_DynamicData_set_float_seq()	424
5.14.4.121 DDS_DynamicData_set_double_seq()	425
5.14.4.122 DDS_DynamicData_set_boolean_seq()	425
5.14.4.123 DDS_DynamicData_set_char_seq()	426
5.14.4.124 DDS_DynamicData_set_octet_seq()	427
5.14.4.125 DDS_DynamicData_set_longlong_seq()	428
5.14.4.126 DDS_DynamicData_set_ulonglong_seq()	428
5.14.4.127 DDS_DynamicData_set_longdouble_seq()	429
5.14.4.128 DDS_DynamicData_set_wchar_seq()	430
5.14.4.129 DDS_DynamicData_set_int8_seq()	431
5.14.4.130 DDS_DynamicData_set_uint8_seq()	431
5.14.4.131 DDS_DynamicDataTypeSupport_new()	432
5.14.4.132 DDS_DynamicDataTypeSupport_delete()	433
5.14.4.133 DDS_DynamicDataTypeSupport_register_type()	433
5.14.4.134 DDS_DynamicDataTypeSupport_unregister_type()	434
5.14.4.135 DDS_DynamicDataTypeSupport_get_type_name()	434
5.14.4.136 DDS_DynamicDataTypeSupport_get_data_type()	435
5.14.4.137 DDS_DynamicDataTypeSupport_create_data()	435
5.14.4.138 DDS_DynamicDataTypeSupport_delete_data()	435
5.14.4.139 DDS_DynamicDataTypeSupport_print_data()	436
5.14.4.140 DDS_DynamicDataTypeSupport_copy_data()	436
5.14.4.141 DDS_DynamicDataTypeSupport_initialize_data()	436
5.14.4.142 DDS_DynamicDataTypeSupport_finalize_data()	437
5.14.5 Variable Documentation	437
5.14.5.1 DDS_DYNAMIC_DATA_PROPERTY_DEFAULT	437
5.14.5.2 DDS_DYNAMIC_DATA_TYPE_PROPERTY_DEFAULT	438
5.14.5.3 DDS_DYNAMIC_DATA_JSON_PARSER_PROPERTIES_DEFAULT	438
5.15 Publication Module	438
5.15.1 Detailed Description	439
5.16 Publishers	439
5.16.1 Detailed Description	442
5.16.2 Macro Definition Documentation	442
5.16.2.1 DDS_PublisherQos_INITIALIZER	442
5.16.2.2 DDS_PublisherListener_INITIALIZER	443
5.16.3 Typedef Documentation	443
5.16.3.1 DDS_Publisher	443
5.16.4 Function Documentation	444
5.16.4.1 DDS_PublisherQos_equals()	444
5.16.4.2 DDS_PublisherQos_print()	444

5.16.4.3 DDS_PublisherQos_to_string()	445
5.16.4.4 DDS_PublisherQos_to_string_w_params()	446
5.16.4.5 DDS_PublisherQos_initialize()	446
5.16.4.6 DDS_PublisherQos_finalize()	447
5.16.4.7 DDS_PublisherQos_copy()	448
5.16.4.8 DDS_Publisher_as_entity()	449
5.16.4.9 DDS_Publisher_get_default_datawriter_qos()	449
5.16.4.10 DDS_Publisher_set_default_datawriter_qos()	450
5.16.4.11 DDS_Publisher_set_default_datawriter_qos_with_profile()	451
5.16.4.12 DDS_Publisher_set_default_profile()	452
5.16.4.13 DDS_Publisher_set_default_library()	452
5.16.4.14 DDS_Publisher_create_datawriter()	453
5.16.4.15 DDS_Publisher_create_datawriter_with_profile()	455
5.16.4.16 DDS_Publisher_delete_datawriter()	456
5.16.4.17 DDS_Publisher_lookup_datawriter()	457
5.16.4.18 DDS_Publisher_suspend_publications()	457
5.16.4.19 DDS_Publisher_resume_publications()	458
5.16.4.20 DDS_Publisher_begin_coherent_changes()	459
5.16.4.21 DDS_Publisher_end_coherent_changes()	460
5.16.4.22 DDS_Publisher_get_all_datawriters()	460
5.16.4.23 DDS_Publisher_get_participant()	461
5.16.4.24 DDS_Publisher_delete_contained_entities()	461
5.16.4.25 DDS_Publisher_copy_from_topic_qos()	462
5.16.4.26 DDS_Publisher_set_qos()	462
5.16.4.27 DDS_Publisher_set_qos_with_profile()	463
5.16.4.28 DDS_Publisher_get_qos()	464
5.16.4.29 DDS_Publisher_get_default_library()	464
5.16.4.30 DDS_Publisher_get_default_profile()	465
5.16.4.31 DDS_Publisher_get_default_profile_library()	465
5.16.4.32 DDS_Publisher_set_listener()	466
5.16.4.33 DDS_Publisher_get_listener()	466
5.16.4.34 DDS_Publisher_get_listenerX()	467
5.16.4.35 DDS_Publisher_wait_for_acknowledgments()	468
5.16.4.36 DDS_Publisher_wait_for_asynchronous_publishing()	468
5.16.4.37 DDS_Publisher_lookup_datawriter_by_name()	469
5.16.5 Variable Documentation	469
5.16.5.1 DDS_DATAWRITER_QOS_PRINT_ALL	470
5.16.5.2 DDS_DATAWRITER_QOS_DEFAULT	470
5.16.5.3 DDS_DATAWRITER_QOS_USE_TOPIC_QOS	471

5.17 Data Writers	471
5.17.1 Detailed Description	479
5.17.2 Macro Definition Documentation	479
5.17.2.1 DDS_OfferedDeadlineMissedStatus_INITIALIZER	480
5.17.2.2 DDS_LivelinessLostStatus_INITIALIZER	480
5.17.2.3 DDS_OfferedIncompatibleQosStatus_INITIALIZER	481
5.17.2.4 DDS_PublicationMatchedStatus_INITIALIZER	481
5.17.2.5 DDS_ServiceRequestAcceptedStatus_INITIALIZER	482
5.17.2.6 DDS_ReliableWriterCacheChangedStatus_INITIALIZER	482
5.17.2.7 DDS_ReliableReaderActivityChangedStatus_INITIALIZER	483
5.17.2.8 DDS_DataWriterCacheStatus_INITIALIZER	483
5.17.2.9 DDS_DataWriterProtocolStatus_INITIALIZER	484
5.17.2.10 DDS_DataWriterQos_INITIALIZER	484
5.17.2.11 DDS_DataWriterListener_INITIALIZER	485
5.17.3 Typedef Documentation	485
5.17.3.1 DDS_DataWriter	485
5.17.3.2 DDS_DataWriterListener_OfferedDeadlineMissedCallback	486
5.17.3.3 DDS_DataWriterListener_LivelinessLostCallback	486
5.17.3.4 DDS_DataWriterListener_OfferedIncompatibleQosCallback	487
5.17.3.5 DDS_DataWriterListener_PublicationMatchedCallback	487
5.17.3.6 DDS_DataWriterListener_ReliableWriterCacheChangedCallback	487
5.17.3.7 DDS_DataWriterListener_ReliableReaderActivityChangedCallback	488
5.17.3.8 DDS_DataWriterListener_SampleRemovedCallback	488
5.17.3.9 DDS_DataWriterListener_InstanceReplacedCallback	489
5.17.3.10 DDS_DataWriterListener_OnApplicationAcknowledgmentCallback	489
5.17.3.11 DDS_DataWriterListener_ServiceRequestAcceptedCallback	489
5.17.4 Function Documentation	490
5.17.4.1 FooDataWriter_narrow()	490
5.17.4.2 FooDataWriter_as_datawriter()	490
5.17.4.3 FooDataWriter_register_instance()	491
5.17.4.4 FooDataWriter_register_instance_w_timestamp()	492
5.17.4.5 FooDataWriter_register_instance_w_params()	493
5.17.4.6 FooDataWriter_unregister_instance()	493
5.17.4.7 FooDataWriter_unregister_instance_w_timestamp()	495
5.17.4.8 FooDataWriter_unregister_instance_w_params()	496
5.17.4.9 FooDataWriter_write()	496
5.17.4.10 FooDataWriter_write_w_timestamp()	500
5.17.4.11 FooDataWriter_write_w_params()	501
5.17.4.12 FooDataWriter_dispose()	502

5.17.4.13 FooDataWriter_dispose_w_timestamp()	504
5.17.4.14 FooDataWriter_dispose_w_params()	505
5.17.4.15 FooDataWriter_get_key_value()	505
5.17.4.16 FooDataWriter_lookup_instance()	506
5.17.4.17 FooDataWriter_create_data()	506
5.17.4.18 FooDataWriter_create_data_w_params()	507
5.17.4.19 FooDataWriter_delete_data()	507
5.17.4.20 FooDataWriter_delete_data_w_params()	508
5.17.4.21 FooDataWriter_get_loan()	509
5.17.4.22 FooDataWriter_discard_loan()	510
5.17.4.23 DDS_OfferedDeadlineMissedStatus_initialize()	510
5.17.4.24 DDS_OfferedDeadlineMissedStatus_copy()	511
5.17.4.25 DDS_OfferedDeadlineMissedStatus_finalize()	512
5.17.4.26 DDS_OfferedDeadlineMissedStatus_equals()	512
5.17.4.27 DDS_LivelinessLostStatus_initialize()	513
5.17.4.28 DDS_LivelinessLostStatus_copy()	514
5.17.4.29 DDS_LivelinessLostStatus_finalize()	514
5.17.4.30 DDS_LivelinessLostStatus_equals()	515
5.17.4.31 DDS_OfferedIncompatibleQosStatus_initialize()	515
5.17.4.32 DDS_OfferedIncompatibleQosStatus_copy()	516
5.17.4.33 DDS_OfferedIncompatibleQosStatus_finalize()	517
5.17.4.34 DDS_OfferedIncompatibleQosStatus_equals()	517
5.17.4.35 DDS_PublicationMatchedStatus_initialize()	518
5.17.4.36 DDS_PublicationMatchedStatus_copy()	518
5.17.4.37 DDS_PublicationMatchedStatus_finalize()	519
5.17.4.38 DDS_PublicationMatchedStatus_equals()	520
5.17.4.39 DDS_ServiceRequestAcceptedStatus_initialize()	520
5.17.4.40 DDS_ServiceRequestAcceptedStatus_copy()	521
5.17.4.41 DDS_ServiceRequestAcceptedStatus_finalize()	521
5.17.4.42 DDS_ServiceRequestAcceptedStatus_equals()	522
5.17.4.43 DDS_ReliableWriterCacheEventCount_equals()	522
5.17.4.44 DDS_ReliableWriterCacheChangedStatus_initialize()	523
5.17.4.45 DDS_ReliableWriterCacheChangedStatus_copy()	524
5.17.4.46 DDS_ReliableWriterCacheChangedStatus_finalize()	524
5.17.4.47 DDS_ReliableWriterCacheChangedStatus_equals()	525
5.17.4.48 DDS_ReliableReaderActivityChangedStatus_initialize()	525
5.17.4.49 DDS_ReliableReaderActivityChangedStatus_copy()	526
5.17.4.50 DDS_ReliableReaderActivityChangedStatus_finalize()	526
5.17.4.51 DDS_ReliableReaderActivityChangedStatus_equals()	528

5.17.4.52 DDS_DataWriterCacheStatus_initialize()	528
5.17.4.53 DDS_DataWriterCacheStatus_copy()	529
5.17.4.54 DDS_DataWriterCacheStatus_finalize()	530
5.17.4.55 DDS_DataWriterCacheStatus_equals()	530
5.17.4.56 DDS_DataWriterProtocolStatus_initialize()	531
5.17.4.57 DDS_DataWriterProtocolStatus_copy()	531
5.17.4.58 DDS_DataWriterProtocolStatus_finalize()	532
5.17.4.59 DDS_DataWriterProtocolStatus_equals()	532
5.17.4.60 DDS_DataWriterQos_equals()	533
5.17.4.61 DDS_DataWriterQos_print()	533
5.17.4.62 DDS_DataWriterQos_to_string()	534
5.17.4.63 DDS_DataWriterQos_to_string_w_params()	535
5.17.4.64 DDS_DataWriterQos_initialize()	535
5.17.4.65 DDS_DataWriterQos_finalize()	536
5.17.4.66 DDS_DataWriterQos_copy()	537
5.17.4.67 DDS_DataWriter_as_entity()	537
5.17.4.68 DDS_DataWriter_assert_liveliness()	538
5.17.4.69 DDS_DataWriter_get_matched_subscription_locators()	538
5.17.4.70 DDS_DataWriter_get_matched_subscriptions()	539
5.17.4.71 DDS_DataWriter_is_matched_subscription_active()	540
5.17.4.72 DDS_DataWriter_get_matched_subscription_data()	541
5.17.4.73 DDS_DataWriter_get_matched_subscription_participant_data()	542
5.17.4.74 DDS_DataWriter_get_topic()	542
5.17.4.75 DDS_DataWriter_get_publisher()	543
5.17.4.76 DDS_DataWriter_wait_for_acknowledgments()	543
5.17.4.77 DDS_DataWriter_is_sample_app_acknowledged()	544
5.17.4.78 DDS_DataWriter_wait_for_asynchronous_publishing()	545
5.17.4.79 DDS_DataWriter_get_liveliness_lost_status()	545
5.17.4.80 DDS_DataWriter_get_offered_deadline_missed_status()	546
5.17.4.81 DDS_DataWriter_get_offered_incompatible_qos_status()	546
5.17.4.82 DDS_DataWriter_get_publication_matched_status()	547
5.17.4.83 DDS_DataWriter_get_reliable_writer_cache_changed_status()	547
5.17.4.84 DDS_DataWriter_get_reliable_reader_activity_changed_status()	547
5.17.4.85 DDS_DataWriter_get_datawriter_cache_status()	548
5.17.4.86 DDS_DataWriter_get_datawriter_protocol_status()	548
5.17.4.87 DDS_DataWriter_get_matched_subscription_datawriter_protocol_status()	549
5.17.4.88 DDS_DataWriter_get_matched_subscription_datawriter_protocol_status_by_locator()	549
5.17.4.89 DDS_DataWriter_get_service_request_accepted_status()	550
5.17.4.90 DDS_DataWriter_set_qos()	551

5.17.4.91 DDS_DataWriter_set_qos_with_profile()	551
5.17.4.92 DDS_DataWriter_get_qos()	552
5.17.4.93 DDS_DataWriter_set_property()	553
5.17.4.94 DDS_DataWriter_set_listener()	553
5.17.4.95 DDS_DataWriter_get_listener()	554
5.17.4.96 DDS_DataWriter_get_listenerX()	554
5.17.4.97 DDS_DataWriter_flush()	555
5.17.4.98 DDS_DataWriter_take_discovery_snapshot()	556
5.18 Flow Controllers	556
5.18.1 Detailed Description	558
5.18.2 Macro Definition Documentation	558
5.18.2.1 DDS_FlowControllerTokenBucketProperty_t_INITIALIZER	558
5.18.3 Typedef Documentation	559
5.18.3.1 DDS_FlowController	559
5.18.4 Enumeration Type Documentation	559
5.18.4.1 DDS_FlowControllerSchedulingPolicy	559
5.18.5 Function Documentation	561
5.18.5.1 DDS_FlowController_get_name()	561
5.18.5.2 DDS_FlowController_get_participant()	562
5.18.5.3 DDS_FlowController_set_property()	562
5.18.5.4 DDS_FlowController_get_property()	563
5.18.5.5 DDS_FlowController_trigger_flow()	563
5.18.6 Variable Documentation	564
5.18.6.1 DDS_DEFAULT_FLOW_CONTROLLER_NAME	564
5.18.6.2 DDS_FIXED_RATE_FLOW_CONTROLLER_NAME	565
5.18.6.3 DDS_ON_DEMAND_FLOW_CONTROLLER_NAME	566
5.19 Subscription Module	567
5.19.1 Detailed Description	567
5.19.2 Access to data samples	567
5.19.2.1 Data access patterns	568
5.20 Subscribers	568
5.20.1 Detailed Description	571
5.20.2 Macro Definition Documentation	571
5.20.2.1 DDS_SubscriberQos_INITIALIZER	571
5.20.2.2 DDS_SubscriberListener_INITIALIZER	572
5.20.3 Typedef Documentation	572
5.20.3.1 DDS_Subscriber	572
5.20.3.2 DDS_SubscriberListener_DataOnReadersCallback	573
5.20.4 Function Documentation	573

5.20.4.1 DDS_SubscriberQos_equals()	574
5.20.4.2 DDS_SubscriberQos_print()	575
5.20.4.3 DDS_SubscriberQos_to_string()	575
5.20.4.4 DDS_SubscriberQos_to_string_w_params()	576
5.20.4.5 DDS_SubscriberQos_initialize()	577
5.20.4.6 DDS_SubscriberQos_finalize()	578
5.20.4.7 DDS_SubscriberQos_copy()	578
5.20.4.8 DDS_Subscriber_as_entity()	579
5.20.4.9 DDS_Subscriber_get_default_datareader_qos()	579
5.20.4.10 DDS_Subscriber_set_default_datareader_qos()	580
5.20.4.11 DDS_Subscriber_set_default_datareader_qos_with_profile()	581
5.20.4.12 DDS_Subscriber_create_datareader()	582
5.20.4.13 DDS_Subscriber_create_datareader_with_profile()	583
5.20.4.14 DDS_Subscriber_delete_datareader()	585
5.20.4.15 DDS_Subscriber_delete_contained_entities()	585
5.20.4.16 DDS_Subscriber_lookup_datareader()	586
5.20.4.17 DDS_Subscriber_begin_access()	587
5.20.4.18 DDS_Subscriber_end_access()	588
5.20.4.19 DDS_Subscriber_get_datareaders()	588
5.20.4.20 DDS_Subscriber_get_all_datareaders()	590
5.20.4.21 DDS_Subscriber_notify_datareaders()	590
5.20.4.22 DDS_Subscriber_get_participant()	591
5.20.4.23 DDS_Subscriber_copy_from_topic_qos()	591
5.20.4.24 DDS_Subscriber_set_qos()	592
5.20.4.25 DDS_Subscriber_set_qos_with_profile()	592
5.20.4.26 DDS_Subscriber_get_qos()	593
5.20.4.27 DDS_Subscriber_set_default_profile()	594
5.20.4.28 DDS_Subscriber_get_default_profile()	595
5.20.4.29 DDS_Subscriber_get_default_profile_library()	595
5.20.4.30 DDS_Subscriber_set_default_library()	596
5.20.4.31 DDS_Subscriber_get_default_library()	596
5.20.4.32 DDS_Subscriber_set_listener()	597
5.20.4.33 DDS_Subscriber_get_listener()	597
5.20.4.34 DDS_Subscriber_get_listenerX()	598
5.20.4.35 DDS_Subscriber_lookup_datareader_by_name()	598
5.20.5 Variable Documentation	600
5.20.5.1 DDS_DATAREADER_QOS_PRINT_ALL	600
5.20.5.2 DDS_DATAREADER_QOS_DEFAULT	600
5.20.5.3 DDS_DATAREADER_QOS_USE_TOPIC_QOS	601

5.21 DataReaders	601
5.21.1 Detailed Description	610
5.21.2 Macro Definition Documentation	610
5.21.2.1 DDS_RequestedDeadlineMissedStatus_INITIALIZER	611
5.21.2.2 DDS_LivelinessChangedStatus_INITIALIZER	611
5.21.2.3 DDS_RequestedIncompatibleQosStatus_INITIALIZER	612
5.21.2.4 DDS_SampleLostStatus_INITIALIZER	612
5.21.2.5 DDS_SampleRejectedStatus_INITIALIZER	613
5.21.2.6 DDS_SubscriptionMatchedStatus_INITIALIZER	613
5.21.2.7 DDS_DataReaderCacheStatus_INITIALIZER	614
5.21.2.8 DDS_DataReaderProtocolStatus_INITIALIZER	614
5.21.2.9 DDS_DataReaderQos_INITIALIZER	615
5.21.2.10 DDS_DataReaderListener_INITIALIZER	615
5.21.3 Typedef Documentation	615
5.21.3.1 DDS_DataReader	616
5.21.3.2 DDS_DataReaderListener_RequestedDeadlineMissedCallback	617
5.21.3.3 DDS_DataReaderListener_LivelinessChangedCallback	617
5.21.3.4 DDS_DataReaderListener_RequestedIncompatibleQosCallback	617
5.21.3.5 DDS_DataReaderListener_SampleRejectedCallback	617
5.21.3.6 DDS_DataReaderListener_DataAvailableCallback	617
5.21.3.7 DDS_DataReaderListener_SubscriptionMatchedCallback	618
5.21.3.8 DDS_DataReaderListener_SampleLostCallback	618
5.21.4 Enumeration Type Documentation	618
5.21.4.1 DDS_SampleLostStatusKind	618
5.21.4.2 DDS_SampleRejectedStatusKind	622
5.21.5 Function Documentation	624
5.21.5.1 FooDataReader_narrow()	624
5.21.5.2 FooDataReader_as_datareader()	624
5.21.5.3 FooDataReader_read()	625
5.21.5.4 FooDataReader_take()	626
5.21.5.5 FooDataReader_read_w_condition()	631
5.21.5.6 FooDataReader_take_w_condition()	632
5.21.5.7 FooDataReader_read_next_sample()	633
5.21.5.8 FooDataReader_take_next_sample()	634
5.21.5.9 FooDataReader_read_instance()	635
5.21.5.10 FooDataReader_take_instance()	636
5.21.5.11 FooDataReader_read_instance_w_condition()	638
5.21.5.12 FooDataReader_take_instance_w_condition()	639
5.21.5.13 FooDataReader_read_next_instance()	640

5.21.5.14 FooDataReader_take_next_instance()	642
5.21.5.15 FooDataReader_read_next_instance_w_condition()	644
5.21.5.16 FooDataReader_take_next_instance_w_condition()	645
5.21.5.17 FooDataReader_return_loan()	646
5.21.5.18 FooDataReader_get_key_value()	647
5.21.5.19 FooDataReader_lookup_instance()	648
5.21.5.20 FooDataReader_is_data_consistent()	649
5.21.5.21 DDS_RequestedDeadlineMissedStatus_initialize()	650
5.21.5.22 DDS_RequestedDeadlineMissedStatus_copy()	650
5.21.5.23 DDS_RequestedDeadlineMissedStatus_finalize()	651
5.21.5.24 DDS_RequestedDeadlineMissedStatus_equals()	651
5.21.5.25 DDS_LivelinessChangedStatus_initialize()	652
5.21.5.26 DDS_LivelinessChangedStatus_copy()	652
5.21.5.27 DDS_LivelinessChangedStatus_finalize()	653
5.21.5.28 DDS_LivelinessChangedStatus_equals()	653
5.21.5.29 DDS_RequestedIncompatibleQosStatus_initialize()	653
5.21.5.30 DDS_RequestedIncompatibleQosStatus_copy()	654
5.21.5.31 DDS_RequestedIncompatibleQosStatus_finalize()	654
5.21.5.32 DDS_RequestedIncompatibleQosStatus_equals()	655
5.21.5.33 DDS_SampleLostStatus_initialize()	655
5.21.5.34 DDS_SampleLostStatus_copy()	656
5.21.5.35 DDS_SampleLostStatus_finalize()	656
5.21.5.36 DDS_SampleLostStatus_equals()	657
5.21.5.37 DDS_SampleRejectedStatus_initialize()	657
5.21.5.38 DDS_SampleRejectedStatus_copy()	658
5.21.5.39 DDS_SampleRejectedStatus_finalize()	658
5.21.5.40 DDS_SampleRejectedStatus_equals()	659
5.21.5.41 DDS_SubscriptionMatchedStatus_initialize()	659
5.21.5.42 DDS_SubscriptionMatchedStatus_copy()	660
5.21.5.43 DDS_SubscriptionMatchedStatus_finalize()	660
5.21.5.44 DDS_SubscriptionMatchedStatus_equals()	661
5.21.5.45 DDS_DataReaderCacheStatus_initialize()	661
5.21.5.46 DDS_DataReaderCacheStatus_copy()	662
5.21.5.47 DDS_DataReaderCacheStatus_finalize()	662
5.21.5.48 DDS_DataReaderCacheStatus_equals()	663
5.21.5.49 DDS_DataReaderProtocolStatus_initialize()	663
5.21.5.50 DDS_DataReaderProtocolStatus_copy()	664
5.21.5.51 DDS_DataReaderProtocolStatus_finalize()	664
5.21.5.52 DDS_DataReaderProtocolStatus_equals()	666

5.21.5.53 DDS_DataReaderQos_equals()	666
5.21.5.54 DDS_DataReaderQos_print()	667
5.21.5.55 DDS_DataReaderQos_to_string()	667
5.21.5.56 DDS_DataReaderQos_to_string_w_params()	668
5.21.5.57 DDS_DataReaderQos_initialize()	669
5.21.5.58 DDS_DataReaderQos_copy()	669
5.21.5.59 DDS_DataReaderQos_finalize()	671
5.21.5.60 DDS_DataReader_as_entity()	671
5.21.5.61 DDS_DataReader_create_readcondition()	671
5.21.5.62 DDS_DataReader_create_readcondition_w_params()	672
5.21.5.63 DDS_DataReader_create_querycondition()	672
5.21.5.64 DDS_DataReader_create_querycondition_w_params()	673
5.21.5.65 DDS_DataReader_delete_readcondition()	674
5.21.5.66 DDS_DataReader_delete_contained_entities()	674
5.21.5.67 DDS_DataReader_wait_for_historical_data()	675
5.21.5.68 DDS_DataReader_acknowledge_sample_w_response()	676
5.21.5.69 DDS_DataReader_acknowledge_all_w_response()	676
5.21.5.70 DDS_DataReader_acknowledge_sample()	677
5.21.5.71 DDS_DataReader_acknowledge_all()	677
5.21.5.72 DDS_DataReader_get_matched_publications()	678
5.21.5.73 DDS_DataReader_is_matched_publication_alive()	679
5.21.5.74 DDS_DataReader_get_matched_publication_data()	680
5.21.5.75 DDS_DataReader_get_matched_publication_participant_data()	680
5.21.5.76 DDS_DataReader_get_topicdescription()	681
5.21.5.77 DDS_DataReader_get_subscriber()	681
5.21.5.78 DDS_DataReader_get_sample_rejected_status()	682
5.21.5.79 DDS_DataReader_get_liveliness_changed_status()	682
5.21.5.80 DDS_DataReader_get_requested_deadline_missed_status()	683
5.21.5.81 DDS_DataReader_get_requested_incompatible_qos_status()	683
5.21.5.82 DDS_DataReader_get_subscription_matched_status()	683
5.21.5.83 DDS_DataReader_get_sample_lost_status()	685
5.21.5.84 DDS_DataReader_get_datareader_cache_status()	685
5.21.5.85 DDS_DataReader_get_datareader_protocol_status()	686
5.21.5.86 DDS_DataReader_get_matched_publication_datareader_protocol_status()	686
5.21.5.87 DDS_DataReader_set_qos()	687
5.21.5.88 DDS_DataReader_set_qos_with_profile()	688
5.21.5.89 DDS_DataReader_get_qos()	688
5.21.5.90 DDS_DataReader_set_property()	689
5.21.5.91 DDS_DataReader_set_listener()	690

5.21.5.92 DDS_DataReader_get_listener()	690
5.21.5.93 DDS_DataReader_get_listenerX()	691
5.21.5.94 DDS_DataReader_create_topic_query()	691
5.21.5.95 DDS_DataReader_delete_topic_query()	692
5.21.5.96 DDS_DataReader_lookup_topic_query()	692
5.21.5.97 DDS_DataReader_take_discovery_snapshot()	693
5.22 Read Conditions	694
5.22.1 Detailed Description	694
5.22.2 Typedef Documentation	694
5.22.2.1 DDS_ReadCondition	695
5.22.3 Function Documentation	695
5.22.3.1 DDS_ReadCondition_as_condition()	695
5.22.3.2 DDS_ReadCondition_get_sample_state_mask()	695
5.22.3.3 DDS_ReadCondition_get_view_state_mask()	696
5.22.3.4 DDS_ReadCondition_get_instance_state_mask()	696
5.22.3.5 DDS_ReadCondition_get_stream_kind_mask()	696
5.22.3.6 DDS_ReadCondition_get_datareader()	696
5.23 Query Conditions	698
5.23.1 Detailed Description	698
5.23.2 Typedef Documentation	699
5.23.2.1 DDS_QueryCondition	699
5.23.3 Function Documentation	699
5.23.3.1 DDS_QueryCondition_as_readcondition()	699
5.23.3.2 DDS_QueryCondition_get_query_expression()	699
5.23.3.3 DDS_QueryCondition_get_query_parameters()	700
5.23.3.4 DDS_QueryCondition_set_query_parameters()	700
5.24 Data Samples	700
5.24.1 Detailed Description	701
5.24.2 Function Documentation	701
5.24.2.1 DDS_CoherentSetInfo_equals()	701
5.24.2.2 DDS_CoherentSetInfo_copy()	702
5.24.2.3 DDS_SampleInfo_get_sample_identity()	702
5.24.2.4 DDS_SampleInfo_get_related_sample_identity()	703
5.25 Topic Queries	703
5.25.1 Detailed Description	704
5.25.2 Snapshot TopicQueries	705
5.25.3 Reading TopicQuery Samples	705
5.25.4 Debugging Topic Queries	706
5.25.4.1 The Built-in ServiceRequest DataReader	706

5.25.4.2 The on_service_request_accepted DataWriter Listener Callback	706
5.25.5 Typedef Documentation	706
5.25.5.1 DDS_TopicQuerySelection	707
5.25.5.2 DDS_TopicQuery	707
5.25.5.3 DDS_TopicQueryData	707
5.25.6 Enumeration Type Documentation	707
5.25.6.1 DDS_TopicQuerySelectionKind	707
5.25.7 Function Documentation	708
5.25.7.1 DDS_TopicQueryHelper_topic_query_data_from_service_request()	708
5.25.7.2 DDS_TopicQuery_get_guid()	709
5.25.8 Variable Documentation	709
5.25.8.1 DDS_TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER	709
5.25.8.2 DDS_TOPIC_QUERY_SELECTION_SELECT_ALL	709
5.26 Sample States	709
5.26.1 Detailed Description	710
5.26.2 Typedef Documentation	710
5.26.2.1 DDS_SampleStateMask	710
5.26.3 Enumeration Type Documentation	710
5.26.3.1 DDS_SampleStateKind	710
5.26.4 Variable Documentation	711
5.26.4.1 DDS_ANY_SAMPLE_STATE	711
5.27 View States	711
5.27.1 Detailed Description	712
5.27.2 Typedef Documentation	712
5.27.2.1 DDS_ViewStateMask	712
5.27.3 Enumeration Type Documentation	712
5.27.3.1 DDS_ViewStateKind	712
5.27.4 Variable Documentation	713
5.27.4.1 DDS_ANY_VIEW_STATE	713
5.28 Instance States	713
5.28.1 Detailed Description	714
5.28.2 Typedef Documentation	714
5.28.2.1 DDS_InstanceStateMask	714
5.28.3 Enumeration Type Documentation	714
5.28.3.1 DDS_InstanceStateKind	714
5.28.4 Variable Documentation	715
5.28.4.1 DDS_ANY_INSTANCE_STATE	715
5.28.4.2 DDS_NOT_ALIVE_INSTANCE_STATE	715
5.29 Stream Kinds	715

5.29.1 Detailed Description	716
5.29.2 Typedef Documentation	716
5.29.2.1 DDS_StreamKindMask	716
5.29.3 Enumeration Type Documentation	716
5.29.3.1 DDS_StreamKind	716
5.30 Infrastructure Module	716
5.30.1 Detailed Description	717
5.31 Built-in Sequences	718
5.31.1 Detailed Description	718
5.32 Multi-channel DataWriters	719
5.32.1 What is a Multi-channel DataWriter?	719
5.32.2 Configuration on the Writer Side	719
5.32.3 Configuration on the Reader Side	719
5.32.4 Reliability with Multi-Channel DataWriters	719
5.32.4.1 Reliable Delivery	719
5.32.4.2 Reliable Protocol Considerations	720
5.33 Transports	720
5.33.1 Detailed Description	721
5.33.2 Overview	721
5.33.3 Transport Aliases	721
5.33.4 Transport Lifecycle	722
5.33.5 Transport Class Attributes	722
5.33.6 Transport Instance Attributes	723
5.33.7 Transport Network Address	724
5.33.8 Transport Send Route	724
5.33.9 Transport Receive Route	725
5.34 Installing Transport Plugins	725
5.34.1 Detailed Description	726
5.34.2 Loading Transport Plugins through Property QoS Policy of Domain Participant	727
5.34.3 Typedef Documentation	729
5.34.3.1 NDDS_Transport_Handle_t	729
5.34.3.2 NDDS_Transport_create_plugin	730
5.34.4 Function Documentation	730
5.34.4.1 NDDS_Transport_Handle_is_nil()	730
5.34.4.2 NDDS_Transport_Support_register_transport()	731
5.34.4.3 NDDS_Transport_Support_lookup_transport()	732
5.34.4.4 NDDS_Transport_Support_add_send_route()	733
5.34.4.5 NDDS_Transport_Support_add_receive_route()	733
5.34.4.6 NDDS_Transport_Support_get_builtin_transport_property()	734

5.34.4.7 NDDS_Transport_Support_set_builtin_transport_property()	735
5.34.4.8 NDDS_Transport_Support_get_transport_plugin()	736
5.34.5 Variable Documentation	736
5.34.5.1 NDDS_TRANSPORT_HANDLE_NIL	736
5.35 Built-in Transport Plugins	736
5.35.1 Detailed Description	737
5.36 Creating New Transport Plugins	737
5.37 Queries and Filters Syntax	738
5.37.1 Syntax for DDS Queries and Filters	738
5.37.2 SQL grammar in BNF	738
5.37.3 Token expression	739
5.37.4 String Parameters	741
5.37.5 Type compatability in Predicate	741
5.37.6 SQL Extension: Regular Expression Matching	742
5.37.7 Character Encoding	743
5.37.8 Unicode Normalization	743
5.37.9 Examples	744
5.38 Logging and Version	744
5.38.1 Detailed Description	744
5.39 General Utilities and Compliance Configuration	744
5.39.1 Detailed Description	745
5.40 Observability Framework	745
5.40.1 Detailed Description	745
5.41 Request-Reply Pattern	745
5.41.1 Detailed Description	745
5.42 Requester	746
5.42.1 Detailed Description	747
5.42.2 Macro Definition Documentation	748
5.42.2.1 RTI_Connext_Requester_DECL	748
5.42.3 Typedef Documentation	748
5.42.3.1 RTI_Connext_RequesterParams	748
5.42.3.2 FooBarRequester	749
5.42.4 Function Documentation	750
5.42.4.1 RTI_Connext_Requester_delete()	750
5.42.4.2 RTI_Connext_Requester_wait_for_replies()	750
5.42.4.3 RTI_Connext_Requester_wait_for_replies_for_related_request()	751
5.42.4.4 FooBarRequester_create()	752
5.42.4.5 FooBarRequester_create_w_params()	752
5.42.4.6 FooBarRequester_send_request()	753

5.42.4.7	FooBarRequester_send_request_w_params()	753
5.42.4.8	FooBarRequester_receive_reply()	754
5.42.4.9	FooBarRequester_receive_replies()	755
5.42.4.10	FooBarRequester_take_reply()	755
5.42.4.11	FooBarRequester_take_replies()	756
5.42.4.12	FooBarRequester_take_reply_for_related_request()	757
5.42.4.13	FooBarRequester_take_replies_for_related_request()	758
5.42.4.14	FooBarRequester_read_reply()	758
5.42.4.15	FooBarRequester_read_replies()	759
5.42.4.16	FooBarRequester_read_reply_for_related_request()	759
5.42.4.17	FooBarRequester_read_replies_for_related_request()	759
5.42.4.18	FooBarRequester_get_request_datawriter()	760
5.42.4.19	FooBarRequester_get_reply_datareader()	760
5.42.4.20	FooBarRequester_return_loan()	761
5.43	Replier	761
5.43.1	Detailed Description	764
5.43.2	Macro Definition Documentation	764
5.43.2.1	RTI_Connext_ReplierListener_INITIALIZER	764
5.43.2.2	RTI_Connext_ReplierParams_INITIALIZER	764
5.43.2.3	RTI_CONNEXT_REPLIER_DECL	764
5.43.2.4	RTI_Connext_RequesterParams_INITIALIZER	765
5.43.2.5	RTI_Connext_SimpleReplierParams_INITIALIZER	765
5.43.2.6	RTI_CONNEXT_SIMPLEREPLIER_DECL	765
5.43.3	Typedef Documentation	765
5.43.3.1	RTI_Connext_ReplierListener_OnRequestAvailableCallback	765
5.43.3.2	RTI_Connext_SimpleReplierListener_OnRequestAvailableCallback	766
5.43.3.3	RTI_Connext_SimpleReplierListener_OnReturnLoanCallback	766
5.43.3.4	RTI_Connext_ReplierParams	766
5.43.3.5	FooBarReplier	766
5.43.3.6	RTI_Connext_SimpleReplierParams	767
5.43.3.7	FooBarSimpleReplier	768
5.43.4	Function Documentation	768
5.43.4.1	RTI_Connext_Replier_delete()	768
5.43.4.2	RTI_Connext_Replier_wait_for_requests()	768
5.43.4.3	FooBarReplier_create()	769
5.43.4.4	FooBarReplier_create_w_params()	769
5.43.4.5	FooBarReplier_receive_request()	770
5.43.4.6	FooBarReplier_receive_requests()	770
5.43.4.7	FooBarReplier_take_request()	771

5.43.4.8 FooBarReplier_take_requests()	771
5.43.4.9 FooBarReplier_read_request()	771
5.43.4.10 FooBarReplier_read_requests()	772
5.43.4.11 FooBarReplier_send_reply()	772
5.43.4.12 FooBarReplier_get_request_datareader()	773
5.43.4.13 FooBarReplier_get_reply_datawriter()	773
5.43.4.14 FooBarReplier_return_loan()	773
5.43.4.15 FooBarSimpleReplier_create()	774
5.43.4.16 FooBarSimpleReplier_create_w_params()	774
5.43.4.17 FooBarSimpleReplier_delete()	774
5.43.5 Variable Documentation	774
5.43.5.1 on_request_available [1/2]	775
5.43.5.2 on_request_available [2/2]	775
5.43.5.3 return_loan	775
5.44 Utilities	776
5.44.1 Detailed Description	776
5.44.2 Typedef Documentation	776
5.44.2.1 RTI_Connext_Messaging_LibraryVersion	776
5.44.3 Function Documentation	776
5.44.3.1 RTI_Connext_Messaging_get_api_version()	777
5.44.3.2 RTI_Connext_Messaging_Library_get_api_version_string()	777
5.44.3.3 RTI_Connext_Messaging_get_api_build_number_string()	777
5.45 Durability and Persistence	777
5.45.1 Durable Writer History	778
5.45.2 Durable Reader State	778
5.45.3 Data Durability	778
5.45.4 Durability and Persistence Based on Virtual GUID	778
5.45.5 Configuring Durable Writer History	779
5.45.6 Configuring Durable Reader State	781
5.45.7 Configuring Data Durability	782
5.46 Basic QoS	783
5.47 DDS Specification QoS	790
5.48 DDS Extension QoS	795
5.49 Functional Categories	833
5.49.1 Detailed Description	835
5.50 Application Acknowledgement	835
5.51 Application-Defined Metadata	836
5.52 Application Startup	838
5.53 Batchting	838

5.54 Concurrency	839
5.55 Content Filtering	841
5.56 Data Availability	843
5.57 Debuggability	847
5.58 Discovery	853
5.59 Discovery Isolation	858
5.60 Durability	860
5.61 Flow Control	863
5.62 Fragmentation	866
5.63 Instance Lifecycle	867
5.64 Sending Large Data	871
5.65 Liveliness	874
5.66 Logging	875
5.67 Middleware Memory Management	878
5.68 QoS Profile Management	893
5.69 Reliability QoS	895
5.70 Sample Memory Management QoS	902
5.71 Sample Ordering	905
5.72 Security	906
5.73 Throughput	908
5.74 Transports	908
5.75 Topic Queries	911
5.76 Type System	912
5.77 QoS Mutability	914
5.77.1 Detailed Description	914
5.78 Immutable QoS	914
5.79 Mutable QoS	949
5.80 Other QoS Attributes	956
5.80.1 Detailed Description	956
5.81 Applied Per-Instance	956
5.82 Discovery Metadata	958
5.83 Matching	963
5.84 Deprecated QoS	965
5.85 System Properties	970
5.85.1 System Properties List	970
5.85.2 System Resource Consideration	971
5.86 Configuring QoS Profiles with XML	971
5.86.1 Loading QoS Profiles from XML Resources	971
5.86.2 URL	973

5.86.2.1 URL groups	973
5.86.2.2 NDDS_QOS_PROFILES environment variable	973
5.86.2.3 Built-In QoS Profiles	973
5.87 Publication Example	973
5.87.1 A typical publication example	974
5.88 Subscription Example	974
5.88.1 A typical subscription example	975
5.89 Participant Use Cases	975
5.89.1 Turning off auto-enable of newly created participant(s)	976
5.89.2 Getting the factory	976
5.89.3 Setting up a participant	976
5.89.4 Tearing down a participant	977
5.90 Topic Use Cases	977
5.90.1 Registering a user data type	978
5.90.2 Setting up a topic	978
5.90.3 Tearing down a topic	978
5.91 FlowController Use Cases	978
5.91.1 Creating a flow controller	979
5.91.2 Flow controlling a data writer	979
5.91.3 Using the built-in flow controllers	979
5.91.4 Shaping the network traffic for a particular transport	980
5.91.5 Coalescing multiple samples in a single network packet	981
5.92 Publisher Use Cases	981
5.92.1 Setting up a publisher	981
5.92.2 Tearing down a publisher	981
5.93 DataWriter Use Cases	982
5.93.1 Setting up a data writer	982
5.93.2 Managing instances	982
5.93.3 Sending data	983
5.93.4 Tearing down a data writer	983
5.94 Subscriber Use Cases	983
5.94.1 Setting up a subscriber	983
5.94.2 Set up subscriber to access received data	984
5.94.3 Access received data via a subscriber	984
5.94.4 Access received data coherently and/or in order	985
5.94.5 Tearing down a subscriber	985
5.95 DataReader Use Cases	985
5.95.1 Setting up a data reader	986
5.95.2 Managing instances	986

5.95.3 Set up reader to access received data	986
5.95.4 Access received data via a reader	987
5.95.5 Taking data	987
5.95.6 Reading data	988
5.95.7 Tearing down a data reader	988
5.96 Entity Use Cases	988
5.96.1 Enabling an entity	988
5.96.2 Checking if a status changed on an entity.	989
5.96.3 Changing the QoS for an entity	989
5.96.4 Changing the listener and enabling/disabling statuses associated with it	989
5.96.5 Enabling/Disabling statuses associated with a status condition	990
5.97 Waitset Use Cases	991
5.97.1 Setting up a wait-set	991
5.97.2 Waiting for condition(s) to trigger	991
5.97.3 Tearing down a wait-set	992
5.98 Transport Use Cases	992
5.98.1 Changing the automatically registered built-in transports	992
5.98.2 Changing the properties of the automatically registered builtin transports	993
5.98.3 Creating a transport	993
5.98.4 Deleting a transport	993
5.98.5 Registering a transport with a participant	994
5.98.6 Adding receive routes for a transport	995
5.98.7 Adding send routes for a transport	995
5.99 Filter Use Cases	995
5.99.1 Introduction	995
5.99.1.1 Overview of ContentFilteredTopic	996
5.99.1.2 Overview of QueryCondition	996
5.99.2 Filtering with ContentFilteredTopic	997
5.99.3 Filtering with Query Conditions	997
5.99.4 Filtering Performance	998
5.100 Creating Custom Content Filters	998
5.100.1 Introduction	999
5.100.2 The Custom Content Filter API	999
5.100.2.1 The compile function	999
5.100.2.2 The evaluate function	999
5.100.2.3 The finalize function	1000
5.100.3 Example Using C format strings	1000
5.100.3.1 Writing the Compile Function	1000
5.100.3.2 Writing the Evaluate Function	1000

5.100.3.3 Writing the Finalize Function	1001
5.100.3.4 Registering the Filter	1001
5.100.3.5 Unregistering the Filter	1001
5.101 Large Data Use Cases	1001
5.101.1 Introduction 	1001
5.101.2 Writing Large Data	1002
5.101.3 Receiving Large Data	1002
5.102 Request-Reply Examples	1002
5.102.1 Request-Reply Examples	1002
5.102.2 Creating a Requester	1004
5.102.3 Creating a Requester with optional parameters	1004
5.102.4 Requester example	1005
5.102.5 Taking loaned samples	1005
5.102.6 Correlating requests and replies	1006
5.102.7 Creating a Replier	1007
5.102.8 Replier example	1008
5.102.9 SimpleReplier example	1009
5.102.10 Configuring Request-Reply QoS profiles	1010
5.103 Documentation Roadmap	1010
5.104 Conventions	1011
5.104.1 Unsupported Features	1011
5.104.2 API Naming Conventions	1011
5.104.2.1 Structure & Class Names	1011
5.104.3 API Documentation Terms	1011
5.104.4 Stereotypes	1012
5.104.4.1 Basic Qos	1012
5.104.4.2 Extensions	1012
5.104.4.3 Experimental	1012
5.104.4.4 Types	1013
5.104.4.5 Method Parameters	1013
5.105 RTI Connex DDS API Reference	1013
5.105.1 Detailed Description	1014
5.105.2 Overview	1015
5.105.3 Conceptual Model	1015
5.105.4 Modules	1017
5.106 Additional RTI Connex Communication Patterns	1017
5.106.1 Detailed Description	1017
5.107 Programming How-To's	1018
5.107.1 Detailed Description	1018

5.108 Interface	1019
5.108.1 Detailed Description	1019
5.108.2 Enumeration Type Documentation	1019
5.108.2.1 NDDS_Transport_Interface_Status_t	1019
5.109 Transport Plugins Configuration	1020
5.109.1 Detailed Description	1021
5.109.2 Macro Definition Documentation	1022
5.109.2.1 NDDS_TRANSPORT_PORT_INVALID	1022
5.109.2.2 NDDS_TRANSPORT_UUID_SIZE	1022
5.109.2.3 NDDS_TRANSPORT_LENGTH_UNLIMITED	1022
5.109.2.4 NDDS_TRANSPORT_INTERFACE_RANK_UNKNOWN	1022
5.109.2.5 NDDS_TRANSPORT_UUID_UNKNOWN	1022
5.109.2.6 NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED	1023
5.109.2.7 NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC	1023
5.109.2.8 NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT	1023
5.109.2.9 NDDS_TRANSPORT_CLASSID_INVALID	1023
5.109.2.10 NDDS_TRANSPORT_CLASSID_UDPv4	1024
5.109.2.11 NDDS_TRANSPORT_CLASSID_SHMEM	1024
5.109.2.12 NDDS_TRANSPORT_CLASSID_SHMEM_510	1024
5.109.2.13 NDDS_TRANSPORT_CLASSID_UDPv6	1024
5.109.2.14 NDDS_TRANSPORT_CLASSID_UDPv6_510	1024
5.109.2.15 NDDS_TRANSPORT_CLASSID_TCPV4_LAN	1024
5.109.2.16 NDDS_TRANSPORT_CLASSID_TCPV4_WAN	1025
5.109.2.17 NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN	1025
5.109.2.18 NDDS_TRANSPORT_CLASSID_TLsv4_LAN	1025
5.109.2.19 NDDS_TRANSPORT_CLASSID_TLsv4_WAN	1025
5.109.2.20 NDDS_TRANSPORT_CLASSID_UDPv4_WAN	1025
5.109.2.21 NDDS_TRANSPORT_CLASSID_RESERVED_RANGE	1025
5.109.2.22 NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED	1026
5.109.2.23 NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_COUNT_MIN	1026
5.109.3 Typedef Documentation	1026
5.109.3.1 NDDS_Transport_Port_t	1026
5.109.3.2 NDDS_Transport_ClassId_t	1027
5.110 Transport Address	1027
5.110.1 Detailed Description	1028
5.110.2 Macro Definition Documentation	1028
5.110.2.1 NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER	1029
5.110.2.2 NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE	1029
5.110.3 Function Documentation	1029

5.110.3.1	NDDS_Transport_Address_to_string()	1029
5.110.3.2	NDDS_Transport_Address_to_string_with_protocol_family_format()	1030
5.110.3.3	NDDS_Transport_Address_from_string()	1031
5.110.3.4	NDDS_Transport_Address_print()	1031
5.110.3.5	NDDS_Transport_Address_is_ipv4()	1031
5.110.3.6	NDDS_Transport_Address_is_multicast()	1032
5.110.4	Variable Documentation	1032
5.110.4.1	NDDS_TRANSPORT_ADDRESS_INVALID	1032
5.111	UDP Transport Plugin definitions	1033
5.111.1	Detailed Description	1033
5.111.2	Macro Definition Documentation	1033
5.111.2.1	NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT	1034
5.111.2.2	NDDS_TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT	1034
5.111.2.3	NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_MAX	1034
5.111.2.4	NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT	1034
5.111.2.5	NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT	1035
5.111.2.6	NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT	1035
5.111.2.7	NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT	1035
5.111.3	Typedef Documentation	1035
5.111.3.1	NDDS_Transport_UDP_Port	1035
5.112	Shared Memory Transport	1035
5.112.1	Detailed Description	1036
5.112.2	Compatibility of Sender and Receiver Transports	1036
5.112.3	Crashing and Restarting Programs	1037
5.112.4	Shared Resource Keys	1037
5.112.5	Creating and Registering Shared Memory Transport Plugin	1037
5.112.6	Shared Memory Transport Property Names in Property QoS Policy of Domain Participant	1038
5.112.7	Macro Definition Documentation	1039
5.112.7.1	NDDS_TRANSPORT_SHMEM_ADDRESS_BIT_COUNT	1039
5.112.7.2	NDDS_TRANSPORT_SHMEM_PROPERTIES_BITMAP_DEFAULT	1039
5.112.7.3	NDDS_TRANSPORT_SHMEM_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT	1039
5.112.7.4	NDDS_TRANSPORT_SHMEM_MESSAGE_SIZE_MAX_DEFAULT	1040
5.112.7.5	NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT	1040
5.112.7.6	NDDS_TRANSPORT_SHMEM_RECEIVE_BUFFER_SIZE_DEFAULT	1040
5.112.7.7	NDDS_TRANSPORT_SHMEM_MAJOR_AFTER_BUG_14240_FIX	1040
5.112.7.8	NDDS_TRANSPORT_SHMEM_PROPERTY_DEFAULT	1040
5.112.8	Function Documentation	1040
5.112.8.1	NDDS_Transport_Shmem_new()	1040
5.112.8.2	NDDS_Transport_Shmem_create()	1041

5.113 UDPv4 Transport	1041
5.113.1 Detailed Description	1043
5.113.2 UDPv4 Transport Property Names in Property QoS Policy of Domain Participant	1044
5.113.3 Macro Definition Documentation	1046
5.113.3.1 NDDS_TRANSPORT_UDPv4_ADDRESS_BIT_COUNT	1046
5.113.3.2 NDDS_TRANSPORT_UDPv4_WAN_ADDRESS_BIT_COUNT	1046
5.113.3.3 NDDS_TRANSPORT_UDPv4_PROPERTIES_BITMAP_DEFAULT	1046
5.113.3.4 NDDS_TRANSPORT_UDPv4_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT	1046
5.113.3.5 NDDS_TRANSPORT_UDPv4_SOCKET_BUFFER_SIZE_OS_MAX	1047
5.113.3.6 NDDS_TRANSPORT_UDPv4_SOCKET_BUFFER_SIZE_OS_DEFAULT	1047
5.113.3.7 NDDS_TRANSPORT_UDPv4_SEND_SOCKET_BUFFER_SIZE_DEFAULT	1047
5.113.3.8 NDDS_TRANSPORT_UDPv4_RECV_SOCKET_BUFFER_SIZE_DEFAULT	1047
5.113.3.9 NDDS_TRANSPORT_UDPv4_PAYLOAD_SIZE_MAX	1047
5.113.3.10 NDDS_TRANSPORT_UDPv4_MESSAGE_SIZE_MAX_DEFAULT	1048
5.113.3.11 NDDS_TRANSPORT_UDPv4_MULTICAST_TTL_DEFAULT	1048
5.113.3.12 NDDS_TRANSPORT_UDPv4_BLOCKING_NEVER	1048
5.113.3.13 NDDS_TRANSPORT_UDPv4_BLOCKING_ALWAYS	1048
5.113.3.14 NDDS_TRANSPORT_UDPv4_BLOCKING_DEFAULT	1048
5.113.3.15 NDDS_TRANSPORT_UDPv4_PROPERTY_DEFAULT	1048
5.113.3.16 NDDS_Transport_UDPv4_string_to_address_cEA	1048
5.113.4 Function Documentation	1049
5.113.4.1 NDDS_Transport_UDPv4_new()	1049
5.113.4.2 NDDS_Transport_UDPv4_create()	1050
5.113.4.3 NDDS_Transport_UDPv4_create_from_properties_with_prefix()	1051
5.114 Real-Time WAN Transport	1051
5.114.1 Detailed Description	1052
5.114.2 Real-Time WAN Transport Property	1053
5.114.3 Macro Definition Documentation	1055
5.114.3.1 NDDS_TRANSPORT_UDPv4_WAN_MESSAGE_SIZE_MAX_DEFAULT	1055
5.114.3.2 NDDS_TRANSPORT_UDPv4_WAN_PROPERTY_DEFAULT	1055
5.114.4 Function Documentation	1056
5.114.4.1 NDDS_Transport_UDPv4_WAN_new()	1056
5.114.4.2 NDDS_Transport_UDPv4_WAN_create()	1056
5.114.4.3 NDDS_Transport_UDPv4_WAN_create_from_properties_with_prefix()	1057
5.115 UDPv6 Transport	1058
5.115.1 Detailed Description	1059
5.115.2 UDPv6 Transport Property Names in Property QoS Policy of Domain Participant	1060
5.115.3 Macro Definition Documentation	1062
5.115.3.1 NDDS_TRANSPORT_UDPv6_ADDRESS_BIT_COUNT	1062

5.115.3.2	NDDS_TRANSPORT_UDPV6_PROPERTIES_BITMAP_DEFAULT	1062
5.115.3.3	NDDS_TRANSPORT_UDPV6_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT	1062
5.115.3.4	NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX	1063
5.115.3.5	NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT	1063
5.115.3.6	NDDS_TRANSPORT_UDPV6_SEND_SOCKET_BUFFER_SIZE_DEFAULT	1063
5.115.3.7	NDDS_TRANSPORT_UDPV6_RECV_SOCKET_BUFFER_SIZE_DEFAULT	1063
5.115.3.8	NDDS_TRANSPORT_UDPV6_PAYLOAD_SIZE_MAX	1063
5.115.3.9	NDDS_TRANSPORT_UDPV6_MESSAGE_SIZE_MAX_DEFAULT	1064
5.115.3.10	NDDS_TRANSPORT_UDPV6_MULTICAST_TTL_DEFAULT	1064
5.115.3.11	NDDS_TRANSPORT_UDPV6_BLOCKING_NEVER	1064
5.115.3.12	NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS	1064
5.115.3.13	NDDS_Transport_UDPv6_string_to_address_cEA	1064
5.115.4	Function Documentation	1065
5.115.4.1	NDDS_Transport_UDPv6_new()	1065
5.115.4.2	NDDS_Transport_UDPv6_create()	1066
5.115.4.3	NDDS_Transport_UDPv6_create_from_properties_with_prefix()	1067
5.116	AsyncWaitSet	1067
5.116.1	Detailed Description	1070
5.116.2	Macro Definition Documentation	1070
5.116.2.1	DDS_AsyncWaitSetListener_INITIALIZER	1070
5.116.3	Typedef Documentation	1070
5.116.3.1	DDS_AsyncWaitSet	1070
5.116.4	AsyncWaitSet Thread Orchestration	1071
5.116.5	AsyncWaitSet Thread Safety	1071
5.116.5.1	Condition Locking	1072
5.116.6	AsyncWaitSet Events and Resources	1072
5.116.6.1	DDS_AsyncWaitSetCompletionToken	1073
5.116.7	AsyncWaitSetCompletionToken management	1073
5.116.7.1	DDS_AsyncWaitSetListener_OnThreadSpawnedCallback	1074
5.116.7.2	DDS_AsyncWaitSetListener_OnThreadDeletedCallback	1074
5.116.7.3	DDS_AsyncWaitSetListener_OnWaitTimeoutCallback	1075
5.116.7.4	DDS_DataReaderStatusConditionHandler	1075
5.116.8	Function Documentation	1076
5.116.8.1	DDS_AsyncWaitSetCompletionToken_wait()	1076
5.116.8.2	DDS_AsyncWaitSet_get_property()	1076
5.116.8.3	DDS_AsyncWaitSet_detach_condition()	1077
5.116.8.4	DDS_AsyncWaitSet_detach_condition_with_completion_token()	1077
5.116.8.5	DDS_AsyncWaitSet_attach_condition()	1078
5.116.8.6	DDS_AsyncWaitSet_attach_condition_with_completion_token()	1079

5.116.8.7 DDS_AsyncWaitSet_unlock_condition()	1080
5.116.8.8 DDS_AsyncWaitSet_get_conditions()	1080
5.116.8.9 DDS_AsyncWaitSet_stop()	1081
5.116.8.10 DDS_AsyncWaitSet_stop_with_completion_token()	1081
5.116.8.11 DDS_AsyncWaitSet_start()	1082
5.116.8.12 DDS_AsyncWaitSet_start_with_completion_token()	1083
5.116.8.13 DDS_AsyncWaitSet_is_started()	1084
5.116.8.14 DDS_AsyncWaitSet_create_completion_token()	1085
5.116.8.15 DDS_AsyncWaitSet_delete_completion_token()	1085
5.116.8.16 DDS_AsyncWaitSet_new()	1086
5.116.8.17 DDS_AsyncWaitSet_new_with_listener()	1086
5.116.8.18 DDS_AsyncWaitSet_new_with_thread_factory()	1088
5.116.8.19 DDS_AsyncWaitSet_delete()	1088
5.116.8.20 DDS_DataReaderStatusConditionHandler_new()	1089
5.116.8.21 DDS_DataReaderStatusConditionHandler_delete()	1090
5.116.9 Variable Documentation	1090
5.116.9.1 DDS_ASYNC_WAITSET_COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT	1090
5.116.9.2 DDS_ASYNC_WAITSET_COMPLETION_TOKEN_IGNORE	1090
5.116.9.3 DDS_ASYNC_WAITSET_PROPERTY_DEFAULT	1091
5.117 Participant Built-in Topics	1091
5.117.1 Detailed Description	1091
5.117.2 Typedef Documentation	1091
5.117.2.1 DDS_ParticipantBuiltinTopicData	1092
5.117.2.2 DDS_ParticipantBuiltinTopicDataDataReader	1092
5.117.3 Variable Documentation	1092
5.117.3.1 DDS_PARTICIPANT_TOPIC_NAME	1092
5.118 Topic Built-in Topics	1093
5.118.1 Detailed Description	1093
5.118.2 Typedef Documentation	1093
5.118.2.1 DDS_TopicBuiltinTopicData	1093
5.118.2.2 DDS_TopicBuiltinTopicDataDataReader	1094
5.118.3 Variable Documentation	1094
5.118.3.1 DDS_TOPIC_TOPIC_NAME	1094
5.119 Publication Built-in Topics	1094
5.119.1 Detailed Description	1095
5.119.2 Typedef Documentation	1095
5.119.2.1 DDS_PublicationBuiltinTopicData	1095
5.119.2.2 DDS_PublicationBuiltinTopicDataDataReader	1096
5.119.3 Variable Documentation	1096

5.119.3.1 DDS_PUBLICATION_TOPIC_NAME	1096
5.120 Subscription Built-in Topics	1096
5.120.1 Detailed Description	1097
5.120.2 Typedef Documentation	1097
5.120.2.1 DDS_SubscriptionBuiltinTopicData	1097
5.120.2.2 DDS_SubscriptionBuiltinTopicDataDataReader	1098
5.120.3 Variable Documentation	1098
5.120.3.1 DDS_SUBSCRIPTION_TOPIC_NAME	1098
5.121 ServiceRequest Built-in Topic	1098
5.121.1 Detailed Description	1099
5.121.2 Typedef Documentation	1100
5.121.2.1 DDS_ServiceRequest	1100
5.121.2.2 DDS_ServiceRequestDataReader	1100
5.121.3 Variable Documentation	1100
5.121.3.1 DDS_UNKNOWN_SERVICE_REQUEST_ID	1101
5.121.3.2 DDS_TOPIC_QUERY_SERVICE_REQUEST_ID	1101
5.121.3.3 DDS_LOCATOR_REACHABILITY_SERVICE_REQUEST_ID	1101
5.121.3.4 DDS_INSTANCE_STATE_SERVICE_REQUEST_ID	1101
5.121.3.5 DDS_MONITORING_LIBRARY_COMMAND_SERVICE_REQUEST_ID	1101
5.121.3.6 DDS_MONITORING_LIBRARY_REPLY_SERVICE_REQUEST_ID	1101
5.121.3.7 DDS_SERVICE_REQUEST_TOPIC_NAME	1102
5.122 Common types and functions	1102
5.122.1 Detailed Description	1104
5.122.2 Macro Definition Documentation	1104
5.122.2.1 DDS_LOCATOR_ADDRESS_LENGTH_MAX	1104
5.122.2.2 DDS_PROTOCOLVERSION_1_0	1105
5.122.2.3 DDS_PROTOCOLVERSION_1_1	1105
5.122.2.4 DDS_PROTOCOLVERSION_1_2	1105
5.122.2.5 DDS_PROTOCOLVERSION_2_0	1105
5.122.2.6 DDS_PROTOCOLVERSION_2_1	1105
5.122.2.7 DDS_PROTOCOLVERSION	1105
5.122.2.8 DDS_VENDOR_ID_LENGTH_MAX	1106
5.122.2.9 DDS_PRODUCTVERSION_UNKNOWN	1106
5.122.2.10 DDS_BuiltinTopicKey_t_INITIALIZER	1106
5.122.3 Typedef Documentation	1106
5.122.3.1 DDS_Locator_t	1106
5.122.3.2 DDS_ProtocolVersion_t	1106
5.122.3.3 DDS_BuiltinTopicKey_t	1107
5.122.4 Function Documentation	1107

5.122.4.1 DDS_BuiltinTopicKey_equals()	1107
5.122.4.2 DDS_BuiltinTopicKey_copy()	1107
5.122.4.3 DDS_BuiltinTopicKey_to_guid()	1108
5.122.4.4 DDS_BuiltinTopicKey_from_guid()	1108
5.122.4.5 DDS_BuiltinTopicKey_to_instance_handle()	1108
5.122.4.6 DDS_BuiltinTopicKey_from_instance_handle()	1109
5.122.5 Variable Documentation	1109
5.122.5.1 DDS_LOCATOR_INVALID	1109
5.122.5.2 DDS_LOCATOR_KIND_INVALID	1109
5.122.5.3 DDS_LOCATOR_PORT_INVALID	1110
5.122.5.4 DDS_LOCATOR_ADDRESS_INVALID	1110
5.122.5.5 DDS_LOCATOR_KIND_UDPv4	1110
5.122.5.6 DDS_LOCATOR_KIND_UDPv4_WAN	1110
5.122.5.7 DDS_LOCATOR_KIND_SHMEM	1110
5.122.5.8 DDS_LOCATOR_KIND_SHMEM_510	1110
5.122.5.9 DDS_LOCATOR_KIND_UDPv6	1111
5.122.5.10 DDS_LOCATOR_KIND_UDPv6_510	1111
5.122.5.11 DDS_LOCATOR_KIND_RESERVED	1111
5.122.5.12 value	1111
5.122.5.13 DDS_BUILTIN_TOPIC_KEY_UNKNOWN	1111
5.123 String Built-in Type	1111
5.123.1 Detailed Description	1113
5.123.2 Typedef Documentation	1114
5.123.2.1 DDS_StringDataWriter	1114
5.123.2.2 DDS_StringDataReader	1114
5.123.3 Function Documentation	1114
5.123.3.1 DDS_StringTypeSupport_register_type()	1114
5.123.3.2 DDS_StringTypeSupport_unregister_type()	1115
5.123.3.3 DDS_StringTypeSupport_get_type_name()	1116
5.123.3.4 DDS_StringTypeSupport_print_data()	1116
5.123.3.5 DDS_StringTypeSupport_get_typecode()	1117
5.123.3.6 DDS_StringTypeSupport_serialize_data_to_cdr_buffer()	1117
5.123.3.7 DDS_StringTypeSupport_serialize_data_to_cdr_buffer_ex()	1117
5.123.3.8 DDS_StringTypeSupport_deserialize_data_from_cdr_buffer()	1118
5.123.3.9 DDS_StringTypeSupport_data_to_string()	1118
5.123.3.10 DDS_StringDataWriter_narrow()	1118
5.123.3.11 DDS_StringDataWriter_as_datawriter()	1119
5.123.3.12 DDS_StringDataWriter_create_data()	1119
5.123.3.13 DDS_StringDataWriter_delete_data()	1119

5.123.3.14 DDS_StringDataWriter_write()	1120
5.123.3.15 DDS_StringDataWriter_write_w_timestamp()	1120
5.123.3.16 DDS_StringDataWriter_write_w_params()	1120
5.123.3.17 DDS_StringDataReader_narrow()	1121
5.123.3.18 DDS_StringDataReader_as_datareader()	1121
5.123.3.19 DDS_StringDataReader_read()	1121
5.123.3.20 DDS_StringDataReader_take()	1122
5.123.3.21 DDS_StringDataReader_read_w_condition()	1122
5.123.3.22 DDS_StringDataReader_take_w_condition()	1122
5.123.3.23 DDS_StringDataReader_read_next_sample()	1123
5.123.3.24 DDS_StringDataReader_take_next_sample()	1123
5.123.3.25 DDS_StringDataReader_return_loan()	1123
5.124 KeyedString Built-in Type	1123
5.124.1 Detailed Description	1128
5.124.2 Typedef Documentation	1129
5.124.2.1 DDS_KeyedString	1129
5.124.2.2 DDS_KeyedStringDataWriter	1129
5.124.2.3 DDS_KeyedStringDataReader	1129
5.124.3 Function Documentation	1129
5.124.3.1 DDS_KeyedString_new()	1130
5.124.3.2 DDS_KeyedString_new_w_size()	1130
5.124.3.3 DDS_KeyedString_delete()	1130
5.124.3.4 DDS_KeyedStringTypeSupport_register_type()	1131
5.124.3.5 DDS_KeyedStringTypeSupport_unregister_type()	1132
5.124.3.6 DDS_KeyedStringTypeSupport_get_type_name()	1133
5.124.3.7 DDS_KeyedStringTypeSupport_print_data()	1133
5.124.3.8 DDS_KeyedStringTypeSupport_get_typecode()	1133
5.124.3.9 DDS_KeyedStringTypeSupport_serialize_data_to_cdr_buffer()	1134
5.124.3.10 DDS_KeyedStringTypeSupport_serialize_data_to_cdr_buffer_ex()	1134
5.124.3.11 DDS_KeyedStringTypeSupport_deserialize_data_from_cdr_buffer()	1134
5.124.3.12 DDS_KeyedStringTypeSupport_data_to_string()	1135
5.124.3.13 DDS_KeyedStringDataWriter_narrow()	1135
5.124.3.14 DDS_KeyedStringDataWriter_as_datawriter()	1135
5.124.3.15 DDS_KeyedStringDataWriter_register_instance()	1136
5.124.3.16 DDS_KeyedStringDataWriter_register_instance_w_key()	1136
5.124.3.17 DDS_KeyedStringDataWriter_register_instance_w_timestamp()	1136
5.124.3.18 DDS_KeyedStringDataWriter_register_instance_w_key_w_timestamp()	1137
5.124.3.19 DDS_KeyedStringDataWriter_unregister_instance()	1137
5.124.3.20 DDS_KeyedStringDataWriter_unregister_instance_w_key()	1137

5.124.3.21 DDS_KeyedStringDataWriter_unregister_instance_w_timestamp()	1138
5.124.3.22 DDS_KeyedStringDataWriter_unregister_instance_w_key_w_timestamp()	1138
5.124.3.23 DDS_KeyedStringDataWriter_create_data()	1138
5.124.3.24 DDS_KeyedStringDataWriter_delete_data()	1139
5.124.3.25 DDS_KeyedStringDataWriter_write()	1139
5.124.3.26 DDS_KeyedStringDataWriter_write_string_w_key()	1139
5.124.3.27 DDS_KeyedStringDataWriter_write_w_timestamp()	1140
5.124.3.28 DDS_KeyedStringDataWriter_write_string_w_key_w_timestamp()	1140
5.124.3.29 DDS_KeyedStringDataWriter_write_w_params()	1140
5.124.3.30 DDS_KeyedStringDataWriter_write_string_w_key_w_params()	1141
5.124.3.31 DDS_KeyedStringDataWriter_dispose()	1141
5.124.3.32 DDS_KeyedStringDataWriter_dispose_w_key()	1141
5.124.3.33 DDS_KeyedStringDataWriter_dispose_w_timestamp()	1142
5.124.3.34 DDS_KeyedStringDataWriter_dispose_w_key_w_timestamp()	1142
5.124.3.35 DDS_KeyedStringDataWriter_get_key_value()	1142
5.124.3.36 DDS_KeyedStringDataWriter_get_key_value_w_key()	1143
5.124.3.37 DDS_KeyedStringDataWriter_lookup_instance()	1143
5.124.3.38 DDS_KeyedStringDataWriter_lookup_instance_w_key()	1143
5.124.3.39 DDS_KeyedStringDataReader_narrow()	1144
5.124.3.40 DDS_KeyedStringDataReader_as_datareader()	1144
5.124.3.41 DDS_KeyedStringDataReader_read()	1144
5.124.3.42 DDS_KeyedStringDataReader_take()	1145
5.124.3.43 DDS_KeyedStringDataReader_read_w_condition()	1145
5.124.3.44 DDS_KeyedStringDataReader_take_w_condition()	1145
5.124.3.45 DDS_KeyedStringDataReader_read_next_sample()	1146
5.124.3.46 DDS_KeyedStringDataReader_take_next_sample()	1146
5.124.3.47 DDS_KeyedStringDataReader_read_instance()	1146
5.124.3.48 DDS_KeyedStringDataReader_take_instance()	1147
5.124.3.49 DDS_KeyedStringDataReader_read_instance_w_condition()	1147
5.124.3.50 DDS_KeyedStringDataReader_take_instance_w_condition()	1147
5.124.3.51 DDS_KeyedStringDataReader_read_next_instance()	1148
5.124.3.52 DDS_KeyedStringDataReader_take_next_instance()	1148
5.124.3.53 DDS_KeyedStringDataReader_read_next_instance_w_condition()	1148
5.124.3.54 DDS_KeyedStringDataReader_take_next_instance_w_condition()	1149
5.124.3.55 DDS_KeyedStringDataReader_return_loan()	1149
5.124.3.56 DDS_KeyedStringDataReader_get_key_value()	1149
5.124.3.57 DDS_KeyedStringDataReader_get_key_value_w_key()	1150
5.124.3.58 DDS_KeyedStringDataReader_lookup_instance()	1150
5.124.3.59 DDS_KeyedStringDataReader_lookup_instance_w_key()	1150

5.124.4 Variable Documentation	1150
5.124.4.1 key	1151
5.124.4.2 value	1151
5.125 Octets Built-in Type	1151
5.125.1 Detailed Description	1154
5.125.2 Typedef Documentation	1154
5.125.2.1 DDS_Octets	1154
5.125.2.2 DDS_OctetsDataWriter	1154
5.125.2.3 DDS_OctetsDataReader	1155
5.125.3 Function Documentation	1155
5.125.3.1 DDS_Octets_new()	1155
5.125.3.2 DDS_Octets_new_w_size()	1155
5.125.3.3 DDS_Octets_delete()	1156
5.125.3.4 DDS_OctetsTypeSupport_register_type()	1156
5.125.3.5 DDS_OctetsTypeSupport_unregister_type()	1157
5.125.3.6 DDS_OctetsTypeSupport_get_type_name()	1158
5.125.3.7 DDS_OctetsTypeSupport_print_data()	1158
5.125.3.8 DDS_OctetsTypeSupport_get_typecode()	1159
5.125.3.9 DDS_OctetsTypeSupport_serialize_data_to_cdr_buffer()	1159
5.125.3.10 DDS_OctetsTypeSupport_serialize_data_to_cdr_buffer_ex()	1159
5.125.3.11 DDS_OctetsTypeSupport_deserialize_data_from_cdr_buffer()	1160
5.125.3.12 DDS_OctetsTypeSupport_data_to_string()	1160
5.125.3.13 DDS_OctetsDataWriter_narrow()	1160
5.125.3.14 DDS_OctetsDataWriter_as_datawriter()	1161
5.125.3.15 DDS_OctetsDataWriter_create_data()	1161
5.125.3.16 DDS_OctetsDataWriter_delete_data()	1161
5.125.3.17 DDS_OctetsDataWriter_write()	1162
5.125.3.18 DDS_OctetsDataWriter_write_octets()	1162
5.125.3.19 DDS_OctetsDataWriter_write_octets_seq()	1162
5.125.3.20 DDS_OctetsDataWriter_write_w_timestamp()	1163
5.125.3.21 DDS_OctetsDataWriter_write_octets_w_timestamp()	1163
5.125.3.22 DDS_OctetsDataWriter_write_octets_seq_w_timestamp()	1164
5.125.3.23 DDS_OctetsDataWriter_write_w_params()	1164
5.125.3.24 DDS_OctetsDataWriter_write_octets_w_params()	1165
5.125.3.25 DDS_OctetsDataWriter_write_octets_seq_w_params()	1165
5.125.3.26 DDS_OctetsDataReader_narrow()	1166
5.125.3.27 DDS_OctetsDataReader_as_datareader()	1166
5.125.3.28 DDS_OctetsDataReader_read()	1166
5.125.3.29 DDS_OctetsDataReader_take()	1167

5.125.3.30 DDS_OctetsDataReader_read_w_condition()	1167
5.125.3.31 DDS_OctetsDataReader_take_w_condition()	1167
5.125.3.32 DDS_OctetsDataReader_read_next_sample()	1168
5.125.3.33 DDS_OctetsDataReader_take_next_sample()	1168
5.125.3.34 DDS_OctetsDataReader_return_loan()	1168
5.125.4 Variable Documentation	1168
5.125.4.1 length	1169
5.125.4.2 value	1169
5.126 KeyedOctets Built-in Type	1169
5.126.1 Detailed Description	1174
5.126.2 Typedef Documentation	1174
5.126.2.1 DDS_KeyedOctets	1175
5.126.2.2 DDS_KeyedOctetsDataWriter	1175
5.126.2.3 DDS_KeyedOctetsDataReader	1175
5.126.3 Function Documentation	1175
5.126.3.1 DDS_KeyedOctets_new()	1176
5.126.3.2 DDS_KeyedOctets_new_w_size()	1176
5.126.3.3 DDS_KeyedOctets_delete()	1176
5.126.3.4 DDS_KeyedOctetsTypeSupport_register_type()	1177
5.126.3.5 DDS_KeyedOctetsTypeSupport_unregister_type()	1178
5.126.3.6 DDS_KeyedOctetsTypeSupport_get_type_name()	1179
5.126.3.7 DDS_KeyedOctetsTypeSupport_print_data()	1179
5.126.3.8 DDS_KeyedOctetsTypeSupport_get_typecode()	1179
5.126.3.9 DDS_KeyedOctetsTypeSupport_serialize_data_to_cdr_buffer()	1180
5.126.3.10 DDS_KeyedOctetsTypeSupport_serialize_data_to_cdr_buffer_ex()	1180
5.126.3.11 DDS_KeyedOctetsTypeSupport_deserialize_data_from_cdr_buffer()	1180
5.126.3.12 DDS_KeyedOctetsTypeSupport_data_to_string()	1181
5.126.3.13 DDS_KeyedOctetsDataWriter_narrow()	1181
5.126.3.14 DDS_KeyedOctetsDataWriter_as_datawriter()	1181
5.126.3.15 DDS_KeyedOctetsDataWriter_register_instance()	1182
5.126.3.16 DDS_KeyedOctetsDataWriter_register_instance_w_key()	1182
5.126.3.17 DDS_KeyedOctetsDataWriter_register_instance_w_timestamp()	1182
5.126.3.18 DDS_KeyedOctetsDataWriter_register_instance_w_key_w_timestamp()	1183
5.126.3.19 DDS_KeyedOctetsDataWriter_unregister_instance()	1183
5.126.3.20 DDS_KeyedOctetsDataWriter_unregister_instance_w_key()	1183
5.126.3.21 DDS_KeyedOctetsDataWriter_unregister_instance_w_timestamp()	1184
5.126.3.22 DDS_KeyedOctetsDataWriter_unregister_instance_w_key_w_timestamp()	1184
5.126.3.23 DDS_KeyedOctetsDataWriter_create_data()	1184
5.126.3.24 DDS_KeyedOctetsDataWriter_delete_data()	1185

5.126.3.25 DDS_KeyedOctetsDataWriter_write()	1185
5.126.3.26 DDS_KeyedOctetsDataWriter_write_octets_w_key()	1185
5.126.3.27 DDS_KeyedOctetsDataWriter_write_octets_seq_w_key()	1186
5.126.3.28 DDS_KeyedOctetsDataWriter_write_w_timestamp()	1186
5.126.3.29 DDS_KeyedOctetsDataWriter_write_octets_w_key_w_timestamp()	1187
5.126.3.30 DDS_KeyedOctetsDataWriter_write_octets_seq_w_key_w_timestamp()	1187
5.126.3.31 DDS_KeyedOctetsDataWriter_write_w_params()	1188
5.126.3.32 DDS_KeyedOctetsDataWriter_write_octets_w_key_w_params()	1188
5.126.3.33 DDS_KeyedOctetsDataWriter_write_octets_seq_w_key_w_params()	1189
5.126.3.34 DDS_KeyedOctetsDataWriter_dispose()	1189
5.126.3.35 DDS_KeyedOctetsDataWriter_dispose_w_key()	1190
5.126.3.36 DDS_KeyedOctetsDataWriter_dispose_w_timestamp()	1190
5.126.3.37 DDS_KeyedOctetsDataWriter_dispose_w_key_w_timestamp()	1190
5.126.3.38 DDS_KeyedOctetsDataWriter_get_key_value()	1191
5.126.3.39 DDS_KeyedOctetsDataWriter_get_key_value_w_key()	1191
5.126.3.40 DDS_KeyedOctetsDataWriter_lookup_instance()	1191
5.126.3.41 DDS_KeyedOctetsDataWriter_lookup_instance_w_key()	1192
5.126.3.42 DDS_KeyedOctetsDataReader_narrow()	1192
5.126.3.43 DDS_KeyedOctetsDataReader_as_datareader()	1192
5.126.3.44 DDS_KeyedOctetsDataReader_read()	1193
5.126.3.45 DDS_KeyedOctetsDataReader_take()	1193
5.126.3.46 DDS_KeyedOctetsDataReader_read_w_condition()	1193
5.126.3.47 DDS_KeyedOctetsDataReader_take_w_condition()	1194
5.126.3.48 DDS_KeyedOctetsDataReader_read_next_sample()	1194
5.126.3.49 DDS_KeyedOctetsDataReader_take_next_sample()	1194
5.126.3.50 DDS_KeyedOctetsDataReader_read_instance()	1195
5.126.3.51 DDS_KeyedOctetsDataReader_take_instance()	1195
5.126.3.52 DDS_KeyedOctetsDataReader_read_instance_w_condition()	1195
5.126.3.53 DDS_KeyedOctetsDataReader_take_instance_w_condition()	1196
5.126.3.54 DDS_KeyedOctetsDataReader_read_next_instance()	1196
5.126.3.55 DDS_KeyedOctetsDataReader_take_next_instance()	1196
5.126.3.56 DDS_KeyedOctetsDataReader_read_next_instance_w_condition()	1197
5.126.3.57 DDS_KeyedOctetsDataReader_take_next_instance_w_condition()	1197
5.126.3.58 DDS_KeyedOctetsDataReader_return_loan()	1197
5.126.3.59 DDS_KeyedOctetsDataReader_get_key_value()	1198
5.126.3.60 DDS_KeyedOctetsDataReader_get_key_value_w_key()	1198
5.126.3.61 DDS_KeyedOctetsDataReader_lookup_instance()	1198
5.126.3.62 DDS_KeyedOctetsDataReader_lookup_instance_w_key()	1199
5.126.4 Variable Documentation	1199

5.126.4.1 key	1199
5.126.4.2 length	1199
5.126.4.3 value	1199
5.127 DDS-Specific Primitive Types	1199
5.127.1 Detailed Description	1201
5.127.2 Macro Definition Documentation	1201
5.127.2.1 DDS_BOOLEAN_TRUE	1201
5.127.2.2 DDS_BOOLEAN_FALSE	1201
5.127.3 Typedef Documentation	1201
5.127.3.1 DDS_Char	1202
5.127.3.2 DDS_Wchar	1202
5.127.3.3 DDS_Octet	1202
5.127.3.4 DDS_UInt8	1202
5.127.3.5 DDS_Int8	1202
5.127.3.6 DDS_Short	1202
5.127.3.7 DDS_UnsignedShort	1203
5.127.3.8 DDS_Long	1203
5.127.3.9 DDS_UnsignedLong	1203
5.127.3.10 DDS_LongLong	1203
5.127.3.11 DDS_UnsignedLongLong	1203
5.127.3.12 DDS_Float	1203
5.127.3.13 DDS_Double	1204
5.127.3.14 DDS_LongDouble	1204
5.127.3.15 DDS_Boolean	1204
5.127.3.16 DDS_Enum	1204
5.128 Time Support	1205
5.128.1 Detailed Description	1206
5.128.2 Macro Definition Documentation	1206
5.128.2.1 DDS_TIME_ZERO	1206
5.128.3 Function Documentation	1206
5.128.3.1 DDS_Time_is_zero()	1206
5.128.3.2 DDS_Time_is_invalid()	1207
5.128.3.3 DDS_Duration_is_infinite()	1207
5.128.3.4 DDS_Duration_is_auto()	1207
5.128.3.5 DDS_Duration_is_zero()	1207
5.128.4 Variable Documentation	1207
5.128.4.1 DDS_TIME_MAX	1208
5.128.4.2 DDS_TIME_INVALID_SEC	1208
5.128.4.3 DDS_TIME_INVALID_NSEC	1208

5.128.4.4 DDS_TIME_INVALID	1208
5.128.4.5 DDS_DURATION_INFINITE_SEC	1208
5.128.4.6 DDS_DURATION_INFINITE_NSEC	1208
5.128.4.7 DDS_DURATION_INFINITE	1209
5.128.4.8 DDS_DURATION_AUTO_SEC	1209
5.128.4.9 DDS_DURATION_AUTO_NSEC	1209
5.128.4.10 DDS_DURATION_AUTO	1209
5.128.4.11 DDS_DURATION_ZERO_SEC	1209
5.128.4.12 DDS_DURATION_ZERO_NSEC	1209
5.128.4.13 DDS_DURATION_ZERO	1210
5.129 GUID Support	1210
5.129.1 Detailed Description	1211
5.129.2 Typedef Documentation	1211
5.129.2.1 DDS_RTPS_GuidPrefix_t	1211
5.129.2.2 DDS_RTPS_EntityId_t	1211
5.129.2.3 DDS_RTPS_GUID_t	1211
5.129.2.4 DDS_GUID_t	1212
5.129.3 Function Documentation	1212
5.129.3.1 DDS_GUID_equals()	1212
5.129.3.2 DDS_GUID_compare()	1212
5.129.3.3 DDS_GUID_copy()	1213
5.129.4 Variable Documentation	1213
5.129.4.1 DDS_GUID_AUTO	1213
5.129.4.2 DDS_GUID_UNKNOWN	1213
5.129.4.3 DDS_GUID_ZERO	1214
5.130 Sequence Number Support	1214
5.130.1 Detailed Description	1215
5.130.2 Typedef Documentation	1215
5.130.2.1 DDS_SequenceNumber_t	1215
5.130.3 Function Documentation	1215
5.130.3.1 DDS_SequenceNumber_subtract()	1215
5.130.3.2 DDS_SequenceNumber_add()	1215
5.130.3.3 DDS_SequenceNumber_plusplus()	1216
5.130.3.4 DDS_SequenceNumber_minusminus()	1216
5.130.3.5 DDS_SequenceNumber_compare()	1216
5.130.4 Variable Documentation	1218
5.130.4.1 DDS_SEQUENCE_NUMBER_UNKNOWN	1218
5.130.4.2 DDS_SEQUENCE_NUMBER_ZERO	1218
5.130.4.3 DDS_SEQUENCE_NUMBER_MAX	1218

5.130.4.4 DDS_AUTO_SEQUENCE_NUMBER	1218
5.131 Exception Codes	1219
5.131.1 Detailed Description	1219
5.131.2 Enumeration Type Documentation	1219
5.131.2.1 DDS_ExceptionCode_t	1219
5.132 Return Codes	1220
5.132.1 Detailed Description	1221
5.132.2 Standard Return Codes	1221
5.132.3 Enumeration Type Documentation	1221
5.132.3.1 DDS_ReturnCode_t	1221
5.133 Status Kinds	1222
5.133.1 Detailed Description	1223
5.133.2 Changes in Status	1224
5.133.2.1 Changes in plain communication status	1225
5.133.2.2 Changes in read communication status	1225
5.133.3 Macro Definition Documentation	1226
5.133.3.1 DDS_STATUS_MASK_NONE	1226
5.133.3.2 DDS_STATUS_MASK_ALL	1226
5.133.4 Typedef Documentation	1226
5.133.4.1 DDS_StatusMask	1227
5.133.5 Enumeration Type Documentation	1227
5.133.5.1 DDS_StatusKind	1227
5.134 Thread Settings	1235
5.134.1 Detailed Description	1236
5.134.2 Macro Definition Documentation	1236
5.134.2.1 DDS_THREAD_SETTINGS_KIND_MASK_DEFAULT	1236
5.134.3 Typedef Documentation	1236
5.134.3.1 DDS_ThreadSettingsKindMask	1236
5.134.4 Enumeration Type Documentation	1237
5.134.4.1 DDS_ThreadSettingsKind	1237
5.134.4.2 DDS_ThreadSettingsCpuRotationKind	1237
5.134.5 Controlling CPU Core Affinity for RTI Threads	1237
5.135 QoS	1238
5.135.1 Detailed Description	1238
5.136 All QoS Policies	1238
5.136.1 Detailed Description	1243
5.136.2 Specifying QoS on entities	1244
5.136.3 QoS compatibility	1245
5.136.4 Macro Definition Documentation	1245

5.136.4.1 DDS_QosPrintFormat_INITIALIZER	1246
5.136.4.2 DDS_QOS_POLICY_COUNT	1246
5.136.5 Enumeration Type Documentation	1246
5.136.5.1 DDS_QosPolicyId_t	1246
5.137 QoS by Category	1249
5.137.1 Detailed Description	1249
5.138 ASYNCHRONOUS_PUBLISHER	1250
5.138.1 Detailed Description	1250
5.138.2 Variable Documentation	1250
5.138.2.1 DDS_ASYNCHRONOUSPUBLISHER_QOS_POLICY_NAME	1250
5.139 AVAILABILITY	1250
5.139.1 Detailed Description	1251
5.139.2 Variable Documentation	1251
5.139.2.1 DDS_AVAILABILITY_QOS_POLICY_NAME	1251
5.140 BATCH	1251
5.140.1 Detailed Description	1251
5.140.2 Variable Documentation	1252
5.140.2.1 DDS_BATCH_QOS_POLICY_NAME	1252
5.141 DATABASE	1252
5.141.1 Detailed Description	1252
5.141.2 Variable Documentation	1252
5.141.2.1 DDS_DATABASE_QOS_POLICY_NAME	1252
5.142 DATA_READER_PROTOCOL	1253
5.142.1 Detailed Description	1253
5.142.2 Variable Documentation	1253
5.142.2.1 DDS_DATAREADERPROTOCOL_QOS_POLICY_NAME	1253
5.143 DATA_READER_RESOURCE_LIMITS	1253
5.143.1 Detailed Description	1254
5.143.2 Enumeration Type Documentation	1254
5.143.2.1 DDS_DataReaderInstanceRemovalKind	1254
5.143.3 Variable Documentation	1256
5.143.3.1 DDS_DATAREADERRESOURCELIMITS_QOS_POLICY_NAME	1256
5.143.3.2 DDS_AUTO_MAX_TOTAL_INSTANCES	1256
5.144 DATA_REPRESENTATION	1256
5.144.1 Detailed Description	1257
5.144.2 Typedef Documentation	1257
5.144.2.1 DDS_DataRepresentationId_t	1257
5.144.3 Variable Documentation	1257
5.144.3.1 DDS_XCDR_DATA_REPRESENTATION	1257

5.144.3.2 DDS_XML_DATA_REPRESENTATION	1258
5.144.3.3 DDS_XCDR2_DATA_REPRESENTATION	1258
5.144.3.4 DDS_AUTO_DATA_REPRESENTATION	1258
5.144.3.5 DDS_DATA_REPRESENTATION_QOS_POLICY_NAME	1258
5.145 Compression Settings	1259
5.145.1 Detailed Description	1260
5.145.2 Macro Definition Documentation	1260
5.145.2.1 DDS_COMPRESSION_ID_MASK_NONE	1260
5.145.2.2 DDS_COMPRESSION_ID_MASK_ALL	1260
5.145.2.3 DDS_COMPRESSION_LEVEL_BEST_COMPRESSION	1260
5.145.2.4 DDS_COMPRESSION_LEVEL_BEST_SPEED	1261
5.145.2.5 DDS_COMPRESSION_LEVEL_DEFAULT	1261
5.145.2.6 DDS_COMPRESSION_THRESHOLD_DEFAULT	1261
5.145.2.7 DDS_COMPRESSION_ID_MASK_PUBLICATION_DEFAULT	1261
5.145.2.8 DDS_COMPRESSION_ID_MASK_SUBSCRIPTION_DEFAULT	1262
5.145.3 Typedef Documentation	1262
5.145.3.1 DDS_CompressionId_t	1262
5.145.3.2 DDS_CompressionIdMask	1262
5.145.4 Variable Documentation	1262
5.145.4.1 DDS_COMPRESSION_ID_ZLIB	1262
5.145.4.2 DDS_COMPRESSION_ID_BZIP2	1263
5.145.4.3 DDS_COMPRESSION_ID_LZ4	1263
5.146 DATA_TAG	1263
5.146.1 Detailed Description	1264
5.146.2 Typedef Documentation	1264
5.146.2.1 DDS_DataTagQosPolicy	1264
5.146.3 Usage	1264
5.146.4 Function Documentation	1265
5.146.4.1 DDS_DataTagQosPolicyHelper_lookup_tag()	1265
5.146.4.2 DDS_DataTagQosPolicyHelper_assert_tag()	1265
5.146.4.3 DDS_DataTagQosPolicyHelper_add_tag()	1266
5.146.4.4 DDS_DataTagQosPolicyHelper_remove_tag()	1267
5.146.4.5 DDS_DataTagQosPolicyHelper_get_number_of_tags()	1267
5.146.5 Variable Documentation	1268
5.146.5.1 DDS_DATATAG_QOS_POLICY_NAME	1268
5.147 DATA_WRITER_PROTOCOL	1268
5.147.1 Detailed Description	1268
5.147.2 Variable Documentation	1268
5.147.2.1 DDS_DATAWRITERPROTOCOL_QOS_POLICY_NAME	1268

5.148 DATA_WRITER_RESOURCE_LIMITS	1269
5.148.1 Detailed Description	1269
5.148.2 Enumeration Type Documentation	1269
5.148.2.1 DDS_DataWriterResourceLimitsInstanceReplacementKind	1270
5.148.3 Variable Documentation	1271
5.148.3.1 DDS_DATAWRITERRESOURCELIMITS_QOS_POLICY_NAME	1272
5.149 DATA_WRITER_TRANSFER_MODE	1272
5.149.1 Detailed Description	1272
5.149.2 Variable Documentation	1272
5.149.2.1 DDS_DATAWRITERTRANSFERMODE_QOS_POLICY_NAME	1272
5.150 DEADLINE	1272
5.150.1 Detailed Description	1273
5.150.2 Variable Documentation	1273
5.150.2.1 DDS_DEADLINE_QOS_POLICY_NAME	1273
5.151 DESTINATION_ORDER	1273
5.151.1 Detailed Description	1274
5.151.2 Enumeration Type Documentation	1274
5.151.2.1 DDS_DestinationOrderQosPolicyKind	1274
5.151.2.2 DDS_DestinationOrderQosPolicyScopeKind	1275
5.151.3 Variable Documentation	1275
5.151.3.1 DDS_DESTINATIONORDER_QOS_POLICY_NAME	1275
5.152 DISCOVERY	1275
5.152.1 Detailed Description	1276
5.152.2 Variable Documentation	1276
5.152.2.1 DDS_DISCOVERY_QOS_POLICY_NAME	1276
5.153 DISCOVERY_CONFIG	1276
5.153.1 Detailed Description	1278
5.153.2 Macro Definition Documentation	1278
5.153.2.1 DDS_DISCOVERYCONFIG_BUILTIN_PLUGIN_MASK_NONE	1278
5.153.2.2 DDS_DISCOVERYCONFIG_BUILTIN_PLUGIN_MASK_DEFAULT	1278
5.153.2.3 DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_NONE	1278
5.153.2.4 DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_DEFAULT	1279
5.153.2.5 DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_ALL	1279
5.153.3 Typedef Documentation	1279
5.153.3.1 DDS_DiscoveryConfigBuiltinPluginKindMask	1279
5.153.3.2 DDS_DiscoveryConfigBuiltinChannelKindMask	1280
5.153.4 Enumeration Type Documentation	1280
5.153.4.1 DDS_DiscoveryConfigBuiltinPluginKind	1280
5.153.4.2 DDS_DiscoveryConfigBuiltinChannelKind	1281

5.153.4.3 DDS_RemoteParticipantPurgeKind	1282
5.153.5 Variable Documentation	1283
5.153.5.1 DDS_DISCOVERYCONFIG_QOS_POLICY_NAME	1283
5.154 DOMAIN_PARTICIPANT_RESOURCE_LIMITS	1283
5.154.1 Detailed Description	1284
5.154.2 Enumeration Type Documentation	1284
5.154.2.1 DDS_DomainParticipantResourceLimitsIgnoredEntityReplacementKind	1284
5.154.3 Variable Documentation	1284
5.154.3.1 DDS_AUTO_COUNT	1285
5.154.3.2 DDS_DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_NAME	1285
5.155 DURABILITY	1285
5.155.1 Detailed Description	1286
5.155.2 Enumeration Type Documentation	1286
5.155.2.1 DDS_PersistentJournalKind	1286
5.155.2.2 DDS_PersistentSynchronizationKind	1287
5.155.2.3 DDS_DurabilityQosPolicyKind	1287
5.155.3 Variable Documentation	1288
5.155.3.1 DDS_DURABILITY_QOS_POLICY_NAME	1288
5.155.3.2 DDS_AUTO_WRITER_DEPTH	1288
5.156 DURABILITY_SERVICE	1289
5.156.1 Detailed Description	1289
5.156.2 Variable Documentation	1289
5.156.2.1 DDS_DURABILITYSERVICE_QOS_POLICY_NAME	1289
5.157 ENTITY_FACTORY	1289
5.157.1 Detailed Description	1290
5.157.2 Variable Documentation	1290
5.157.2.1 DDS_ENTITYFACTORY_QOS_POLICY_NAME	1290
5.158 ENTITY_NAME	1290
5.158.1 Detailed Description	1291
5.158.2 Variable Documentation	1291
5.158.2.1 DDS_ENTITYNAME_QOS_POLICY_NAME	1291
5.159 EVENT	1291
5.159.1 Detailed Description	1291
5.159.2 Variable Documentation	1292
5.159.2.1 DDS_EVENT_QOS_POLICY_NAME	1292
5.160 HISTORY	1292
5.160.1 Detailed Description	1292
5.160.2 Enumeration Type Documentation	1292
5.160.2.1 DDS_HistoryQosPolicyKind	1292

5.160.3 Variable Documentation	1293
5.160.3.1 DDS_HISTORY_QOS_POLICY_NAME	1293
5.161 GROUP_DATA	1293
5.161.1 Detailed Description	1294
5.161.2 Variable Documentation	1294
5.161.2.1 DDS_GROUPDATA_QOS_POLICY_NAME	1294
5.162 LATENCY_BUDGET	1294
5.162.1 Detailed Description	1294
5.162.2 Variable Documentation	1295
5.162.2.1 DDS_LATENCYBUDGET_QOS_POLICY_NAME	1295
5.163 LIFESPAN	1295
5.163.1 Detailed Description	1295
5.163.2 Variable Documentation	1295
5.163.2.1 DDS_LIFESPAN_QOS_POLICY_NAME	1295
5.164 LIVELINESS	1296
5.164.1 Detailed Description	1296
5.164.2 Enumeration Type Documentation	1296
5.164.2.1 DDS_LivelinessQosPolicyKind	1296
5.164.3 Variable Documentation	1297
5.164.3.1 DDS_LIVELINESS_QOS_POLICY_NAME	1297
5.165 LOCATORFILTER	1297
5.165.1 Detailed Description	1298
5.165.2 Variable Documentation	1298
5.165.2.1 DDS_LOCATORFILTER_QOS_POLICY_NAME	1298
5.166 LOGGING	1298
5.166.1 Detailed Description	1299
5.167 MONITORING	1299
5.167.1 Detailed Description	1299
5.167.2 Variable Documentation	1300
5.167.2.1 DDS_MONITORING_QOS_POLICY_NAME	1300
5.168 MULTICHANNEL	1300
5.168.1 Detailed Description	1300
5.168.2 Variable Documentation	1300
5.168.2.1 DDS_MULTICHANNEL_QOS_POLICY_NAME	1301
5.169 OWNERSHIP	1301
5.169.1 Detailed Description	1301
5.169.2 Enumeration Type Documentation	1301
5.169.2.1 DDS_OwnershipQosPolicyKind	1301
5.169.3 Variable Documentation	1302

5.169.3.1 DDS_OWNERSHIP_QOS_POLICY_NAME	1302
5.170 OWNERSHIP_STRENGTH	1302
5.170.1 Detailed Description	1303
5.170.2 Variable Documentation	1303
5.170.2.1 DDS_OWNERSHIPSTRENGTH_QOS_POLICY_NAME	1303
5.171 PARTITION	1303
5.171.1 Detailed Description	1303
5.171.2 Variable Documentation	1303
5.171.2.1 DDS_PARTITION_QOS_POLICY_NAME	1304
5.172 PRESENTATION	1304
5.172.1 Detailed Description	1304
5.172.2 Enumeration Type Documentation	1304
5.172.2.1 DDS_PresentationQosPolicyAccessScopeKind	1304
5.172.3 Variable Documentation	1305
5.172.3.1 DDS_PRESENTATION_QOS_POLICY_NAME	1305
5.173 PROFILE	1305
5.173.1 Detailed Description	1306
5.173.2 Variable Documentation	1306
5.173.2.1 DDS_PROFILE_QOS_POLICY_NAME	1306
5.174 PROPERTY	1306
5.174.1 Detailed Description	1308
5.174.2 Typedef Documentation	1308
5.174.2.1 DDS_PropertyQosPolicy	1308
5.174.3 Usage	1308
5.174.3.1 Reasons for Using the PropertyQosPolicy	1309
5.174.3.2 DDS_PropertyQosPolicyMutability	1309
5.174.4 Enumeration Type Documentation	1309
5.174.4.1 DDS_PropertyQosPolicyMutability	1309
5.174.5 Function Documentation	1310
5.174.5.1 DDS_PropertyQosPolicyHelper_get_number_of_properties()	1310
5.174.5.2 DDS_PropertyQosPolicyHelper_assert_property()	1310
5.174.5.3 DDS_PropertyQosPolicyHelper_add_property()	1311
5.174.5.4 DDS_PropertyQosPolicyHelper_assert_pointer_property()	1312
5.174.5.5 DDS_PropertyQosPolicyHelper_add_pointer_property()	1312
5.174.5.6 DDS_PropertyQosPolicyHelper_lookup_property()	1313
5.174.5.7 DDS_PropertyQosPolicyHelper_lookup_property_with_prefix()	1314
5.174.5.8 DDS_PropertyQosPolicyHelper_remove_property()	1314
5.174.5.9 DDS_PropertyQosPolicyHelper_get_properties()	1315
5.174.5.10 DDS_PropertyQosPolicyHelper_get_properties_into_policy()	1316

5.174.5.11 DDS_PropertyQosPolicyHelper_get_property_mutability()	1316
5.174.6 Variable Documentation	1317
5.174.6.1 DDS_PROPERTY_QOS_POLICY_NAME	1317
5.175 PUBLISH_MODE	1317
5.175.1 Detailed Description	1318
5.175.2 Macro Definition Documentation	1318
5.175.2.1 DDS_PUBLICATION_PRIORITY_UNDEFINED	1318
5.175.2.2 DDS_PUBLICATION_PRIORITY_AUTOMATIC	1318
5.175.3 Enumeration Type Documentation	1318
5.175.3.1 DDS_PublishModeQosPolicyKind	1318
5.175.4 Variable Documentation	1319
5.175.4.1 DDS_PUBLISHMODE_QOS_POLICY_NAME	1320
5.176 READER_DATA_LIFECYCLE	1320
5.176.1 Detailed Description	1320
5.176.2 Variable Documentation	1320
5.176.2.1 DDS_READERDATALIFECYCLE_QOS_POLICY_NAME	1320
5.177 RECEIVER_POOL	1320
5.177.1 Detailed Description	1321
5.177.2 Variable Documentation	1321
5.177.2.1 DDS_RECEIVERPOOL_QOS_POLICY_NAME	1321
5.177.2.2 DDS_LENGTH_AUTO	1321
5.178 RELIABILITY	1321
5.178.1 Detailed Description	1322
5.178.2 Enumeration Type Documentation	1322
5.178.2.1 DDS_ReliabilityQosPolicyKind	1322
5.178.2.2 DDS_ReliabilityQosPolicyAcknowledgmentModeKind	1323
5.178.2.3 DDS_InstanceStateConsistencyKind	1324
5.178.3 Variable Documentation	1325
5.178.3.1 DDS_RELIABILITY_QOS_POLICY_NAME	1325
5.179 RESOURCE_LIMITS	1325
5.179.1 Detailed Description	1325
5.179.2 Variable Documentation	1325
5.179.2.1 DDS_RESOURCELIMITS_QOS_POLICY_NAME	1326
5.179.2.2 DDS_LENGTH_UNLIMITED	1326
5.180 SERVICE	1326
5.180.1 Detailed Description	1327
5.180.2 Enumeration Type Documentation	1327
5.180.2.1 DDS_ServiceQosPolicyKind	1327
5.180.3 Variable Documentation	1328

5.180.3.1 DDS_SERVICE_QOS_POLICY_NAME	1328
5.181 SYSTEM_RESOURCE_LIMITS	1328
5.181.1 Detailed Description	1328
5.181.2 Variable Documentation	1328
5.181.2.1 DDS_SYSTEMRESOURCELIMITS_QOS_POLICY_NAME	1328
5.182 TIME_BASED_FILTER	1329
5.182.1 Detailed Description	1329
5.182.2 Variable Documentation	1329
5.182.2.1 DDS_TIMEBASEDFILTER_QOS_POLICY_NAME	1329
5.183 TOPIC_DATA	1329
5.183.1 Detailed Description	1330
5.183.2 Variable Documentation	1330
5.183.2.1 DDS_TOPICDATA_QOS_POLICY_NAME	1330
5.184 TOPIC_QUERY_DISPATCH	1330
5.184.1 Detailed Description	1330
5.184.2 Variable Documentation	1331
5.184.2.1 DDS_TOPICQUERYDISPATCH_QOS_POLICY_NAME	1331
5.185 TRANSPORT_BUILTIN	1331
5.185.1 Detailed Description	1332
5.185.2 Macro Definition Documentation	1332
5.185.2.1 DDS_TRANSPORTBUILTIN_MASK_NONE	1332
5.185.2.2 DDS_TRANSPORTBUILTIN_MASK_DEFAULT	1333
5.185.2.3 DDS_TRANSPORTBUILTIN_MASK_ALL	1333
5.185.3 Typedef Documentation	1333
5.185.3.1 DDS_TransportBuiltinKindMask	1333
5.185.4 Enumeration Type Documentation	1333
5.185.4.1 DDS_TransportBuiltinKind	1333
5.185.5 Variable Documentation	1334
5.185.5.1 DDS_TRANSPORTBUILTIN_QOS_POLICY_NAME	1334
5.185.5.2 DDS_TRANSPORTBUILTIN_SHMEM_ALIAS	1334
5.185.5.3 DDS_TRANSPORTBUILTIN_UDPv4_ALIAS	1334
5.185.5.4 DDS_TRANSPORTBUILTIN_UDPv4_WAN_ALIAS	1334
5.185.5.5 DDS_TRANSPORTBUILTIN_UDPv6_ALIAS	1335
5.186 TRANSPORT_MULTICAST	1335
5.186.1 Detailed Description	1335
5.186.2 Enumeration Type Documentation	1336
5.186.2.1 DDS_TransportMulticastQosPolicyKind	1336
5.186.3 Variable Documentation	1336
5.186.3.1 DDS_TRANSPORTMULTICAST_QOS_POLICY_NAME	1336

5.187	TRANSPORT_MULTICAST_MAPPING	1336
5.187.1	Detailed Description	1337
5.187.2	Variable Documentation	1337
5.187.2.1	DDS_TRANSPORTMULTICASTMAPPING_QOS_POLICY_NAME	1337
5.188	TRANSPORT_PRIORITY	1337
5.188.1	Detailed Description	1337
5.188.2	Variable Documentation	1338
5.188.2.1	DDS_TRANSPORTPRIORITY_QOS_POLICY_NAME	1338
5.189	TRANSPORT_SELECTION	1338
5.189.1	Detailed Description	1338
5.189.2	Variable Documentation	1338
5.189.2.1	DDS_TRANSPORTSELECTION_QOS_POLICY_NAME	1338
5.190	TRANSPORT_UNICAST	1339
5.190.1	Detailed Description	1339
5.190.2	Variable Documentation	1339
5.190.2.1	DDS_TRANSPORTUNICAST_QOS_POLICY_NAME	1339
5.191	TYPE_CONSISTENCY_ENFORCEMENT	1339
5.191.1	Detailed Description	1340
5.191.2	Enumeration Type Documentation	1340
5.191.2.1	DDS_TypeConsistencyKind	1340
5.191.3	Variable Documentation	1341
5.191.3.1	DDS_TYPE_CONSISTENCY_ENFORCEMENT_QOS_POLICY_NAME	1341
5.192	TYPESUPPORT	1341
5.192.1	Detailed Description	1342
5.192.2	Enumeration Type Documentation	1342
5.192.2.1	DDS_CdrPaddingKind	1342
5.192.3	Variable Documentation	1343
5.192.3.1	DDS_TYPESUPPORT_QOS_POLICY_NAME	1343
5.193	USER_DATA	1343
5.193.1	Detailed Description	1343
5.193.2	Variable Documentation	1343
5.193.2.1	DDS_USERDATA_QOS_POLICY_NAME	1343
5.194	WRITER_DATA_LIFECYCLE	1344
5.194.1	Detailed Description	1344
5.194.2	Variable Documentation	1344
5.194.2.1	DDS_WRITERDATALIFECYCLE_QOS_POLICY_NAME	1344
5.195	WIRE_PROTOCOL	1344
5.195.1	Detailed Description	1345
5.195.2	Macro Definition Documentation	1345

5.195.2.1 DDS_RTPS_RESERVED_PORT_MASK_DEFAULT	1346
5.195.2.2 DDS_RTPS_RESERVED_PORT_MASK_NONE	1346
5.195.2.3 DDS_RTPS_RESERVED_PORT_MASK_ALL	1347
5.195.3 Typedef Documentation	1347
5.195.3.1 DDS_RtpsReservedPortKindMask	1347
5.195.4 Enumeration Type Documentation	1347
5.195.4.1 DDS_RtpsReservedPortKind	1347
5.195.4.2 DDS_WireProtocolQosPolicyAutoKind	1348
5.195.5 Variable Documentation	1348
5.195.5.1 DDS_RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS	1348
5.195.5.2 DDS_INTEROPERABLE_RTPS_WELL_KNOWN_PORTS	1349
5.195.5.3 DDS_WIREPROTOCOL_QOS_POLICY_NAME	1350
5.196 Extended Qos Support	1350
5.196.1 Detailed Description	1350
5.197 Unicast Settings	1350
5.197.1 Detailed Description	1350
5.198 Multicast Settings	1351
5.198.1 Detailed Description	1351
5.199 Multicast Mapping	1351
5.199.1 Detailed Description	1351
5.200 NDDS_DISCOVERY_PEERS	1352
5.200.1 Peer Descriptor Format	1353
5.200.1.1 Locator Format	1353
5.200.1.2 Address Format	1354
5.200.2 NDDS_DISCOVERY_PEERS Environment Variable Format	1354
5.200.3 NDDS_DISCOVERY_PEERS File Format	1356
5.200.4 NDDS_DISCOVERY_PEERS Precedence	1357
5.200.5 NDDS_DISCOVERY_PEERS Default Value	1357
5.200.6 Builtin Transport Class Names	1358
5.200.7 NDDS_DISCOVERY_PEERS and Local Host Communication	1358
5.201 Entity Support	1359
5.201.1 Detailed Description	1360
5.201.2 Macro Definition Documentation	1360
5.201.2.1 DDS_Listener_INITIALIZER	1360
5.201.3 Typedef Documentation	1360
5.201.3.1 DDS_Entity	1361
5.201.4 Abstract operations	1361
5.201.4.1 set_qos (abstract)	1361
5.201.4.2 get_qos (abstract)	1362

5.201.4.3 set_listener (abstract)	1362
5.201.4.4 get_listener (abstract)	1363
5.201.4.5 DDS_DomainEntity	1363
5.201.5 Function Documentation	1363
5.201.5.1 DDS_Entity_enable()	1364
5.201.5.2 DDS_Entity_get_statuscondition()	1365
5.201.5.3 DDS_Entity_get_status_changes()	1365
5.201.5.4 DDS_Entity_get_instance_handle()	1366
5.201.5.5 DDS_Entity_get_entity_kind()	1366
5.202 Conditions and WaitSets	1367
5.202.1 Detailed Description	1369
5.202.2 Macro Definition Documentation	1369
5.202.2.1 DDS_ConditionHandler_INITIALIZER	1369
5.202.2.2 DDS_WaitSetProperty_t_INITIALIZER	1369
5.202.3 Typedef Documentation	1370
5.202.3.1 DDS_Condition	1370
5.202.3.2 DDS_ConditionHandler_OnConditionTriggeredCallback	1370
5.202.3.3 DDS_GuardCondition	1370
5.202.3.4 DDS_StatusCondition	1371
5.202.3.5 DDS_WaitSet	1371
5.202.4 Usage	1371
5.202.5 Trigger State of a ::DDS_StatusCondition	1372
5.202.6 Trigger State of a ::DDS_ReadCondition	1373
5.202.7 Trigger State of a ::DDS_GuardCondition	1373
5.202.8 Function Documentation	1373
5.202.8.1 DDS_Condition_get_trigger_value()	1374
5.202.8.2 DDS_Condition_set_handler()	1375
5.202.8.3 DDS_Condition_get_handler()	1375
5.202.8.4 DDS_Condition_dispatch()	1376
5.202.8.5 DDS_GuardCondition_as_condition()	1376
5.202.8.6 DDS_GuardCondition_new()	1376
5.202.8.7 DDS_GuardCondition_delete()	1377
5.202.8.8 DDS_GuardCondition_set_trigger_value()	1377
5.202.8.9 DDS_StatusCondition_as_condition()	1377
5.202.8.10 DDS_StatusCondition_get_enabled_statuses()	1379
5.202.8.11 DDS_StatusCondition_set_enabled_statuses()	1379
5.202.8.12 DDS_StatusCondition_get_entity()	1380
5.202.8.13 DDS_WaitSet_new()	1380
5.202.8.14 DDS_WaitSet_new_ex()	1381

5.202.8.15 DDS_WaitSet_delete()	1381
5.202.8.16 DDS_WaitSet_set_property()	1382
5.202.8.17 DDS_WaitSet_get_property()	1382
5.202.8.18 DDS_WaitSet_wait()	1383
5.202.8.19 DDS_WaitSet_attach_condition()	1383
5.202.8.20 DDS_WaitSet_detach_condition()	1384
5.202.8.21 DDS_WaitSet_get_conditions()	1385
5.203 Cookie	1385
5.203.1 Detailed Description	1385
5.203.2 Function Documentation	1385
5.203.2.1 DDS_Cookie_from_pointer()	1386
5.203.2.2 DDS_Cookie_to_pointer()	1386
5.204 Sample Flags	1386
5.204.1 Detailed Description	1387
5.204.2 Typedef Documentation	1387
5.204.2.1 DDS_SampleFlagBits	1387
5.204.2.2 DDS_SampleFlag	1387
5.204.3 Enumeration Type Documentation	1388
5.204.3.1 DDS_SampleFlagBits	1388
5.205 WriteParams	1389
5.205.1 Detailed Description	1389
5.205.2 Function Documentation	1389
5.205.2.1 DDS_SampleIdentity_equals()	1390
5.205.2.2 DDS_WriteParams_reset()	1391
5.205.3 Variable Documentation	1391
5.205.3.1 writer_guid	1391
5.205.3.2 sequence_number	1391
5.205.3.3 DDS_AUTO_SAMPLE_IDENTITY	1392
5.205.3.4 DDS_UNKNOWN_SAMPLE_IDENTITY	1392
5.205.3.5 DDS_WRITEPARAMS_DEFAULT	1392
5.206 Heap Support in C	1392
5.206.1 Detailed Description	1392
5.206.2 Function Documentation	1393
5.206.2.1 DDS_Heap_calloc()	1393
5.206.2.2 DDS_Heap_malloc()	1393
5.206.2.3 DDS_Heap_free()	1394
5.207 Builtin Qos Profiles	1394
5.207.1 Detailed Description	1398
5.207.2 Variable Documentation	1400

5.207.2.1 DDS_BUILTIN_QOS_LIB	1400
5.207.2.2 DDS_PROFILE_BASELINE_ROOT	1400
5.207.2.3 DDS_PROFILE_BASELINE	1400
5.207.2.4 DDS_PROFILE_BASELINE_5_0_0	1400
5.207.2.5 DDS_PROFILE_BASELINE_5_1_0	1401
5.207.2.6 DDS_PROFILE_BASELINE_5_2_0	1401
5.207.2.7 DDS_PROFILE_BASELINE_5_3_0	1401
5.207.2.8 DDS_PROFILE_BASELINE_6_0_0	1401
5.207.2.9 DDS_PROFILE_BASELINE_6_1_0	1401
5.207.2.10 DDS_PROFILE_BASELINE_7_0_0	1402
5.207.2.11 DDS_PROFILE_BASELINE_7_1_0	1402
5.207.2.12 DDS_PROFILE_GENERIC_COMMON	1402
5.207.2.13 DDS_PROFILE_GENERIC_MONITORING_COMMON	1402
5.207.2.14 DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY	1403
5.207.2.15 DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_2_4_9	1403
5.207.2.16 DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_2_4_3	1403
5.207.2.17 DDS_PROFILE_GENERIC_OTHER_DDS_VENDOR_COMPATIBILITY	1403
5.207.2.18 DDS_PROFILE_GENERIC_510_TRANSPORT_COMPATIBILITY	1404
5.207.2.19 DDS_BUILTIN_QOS_LIB_EXP	1404
5.207.2.20 DDS_PROFILE_GENERIC_STRICT_RELIABLE	1404
5.207.2.21 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE	1404
5.207.2.22 DDS_PROFILE_GENERIC_BEST_EFFORT	1405
5.207.2.23 DDS_PROFILE_GENERIC_STRICT_RELIABLE_HIGH_THROUGHPUT	1405
5.207.2.24 DDS_PROFILE_GENERIC_STRICT_RELIABLE_LOW_LATENCY	1405
5.207.2.25 DDS_PROFILE_GENERIC_PARTICIPANT_LARGE_DATA	1406
5.207.2.26 DDS_PROFILE_GENERIC_PARTICIPANT_LARGE_DATA_MONITORING	1406
5.207.2.27 DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA	1407
5.207.2.28 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA	1407
5.207.2.29 DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_FAST_FLOW	1407
5.207.2.30 DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_MEDIUM_FLOW	1408
5.207.2.31 DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_SLOW_FLOW	1408
5.207.2.32 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_FAST_FLOW	1408
5.207.2.33 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_MEDIUM_FLOW	1408
5.207.2.34 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_SLOW_FLOW	1409
5.207.2.35 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT_LOCAL	1409
5.207.2.36 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT	1409
5.207.2.37 DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_PERSISTENT	1409
5.207.2.38 DDS_PROFILE_GENERIC_AUTO_TUNING	1410
5.207.2.39 DDS_PROFILE_GENERIC_MINIMAL_MEMORY_FOOTPRINT	1410

5.207.2.40 DDS_PROFILE_GENERIC_SECURITY	1410
5.207.2.41 DDS_PROFILE_GENERIC_MONITORING2	1411
5.207.2.42 DDS_PROFILE_GENERIC_MONITORING2_WAN	1412
5.207.2.43 DDS_PROFILE_PATTERN_RPC	1412
5.207.2.44 DDS_PROFILE_PATTERN_PERIODIC_DATA	1412
5.207.2.45 DDS_PROFILE_PATTERN_STREAMING	1413
5.207.2.46 DDS_PROFILE_PATTERN_RELIABLE_STREAMING	1413
5.207.2.47 DDS_PROFILE_PATTERN_EVENT	1413
5.207.2.48 DDS_PROFILE_PATTERN_ALARM_EVENT	1414
5.207.2.49 DDS_PROFILE_PATTERN_STATUS	1414
5.207.2.50 DDS_PROFILE_PATTERN_ALARM_STATUS	1414
5.207.2.51 DDS_PROFILE_PATTERN_LAST_VALUE_CACHE	1415
5.207.2.52 DDS_BUILTIN_QOS_SNIPPET_LIB	1415
5.207.2.53 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_COMMON	1415
5.207.2.54 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_ALL	1416
5.207.2.55 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_LAST	1416
5.207.2.56 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_HIGH_RATE	1417
5.207.2.57 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LOW_LATENCY	1417
5.207.2.58 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LARGE_DATA	1418
5.207.2.59 DDS_SNIPPET_OPTIMIZATION_DATACACHE_LARGE_DATA_DYNAMICMEMALLOC	1418
5.207.2.60 DDS_SNIPPET_OPTIMIZATION_DISCOVERY_COMMON	1419
5.207.2.61 DDS_SNIPPET_OPTIMIZATION_DISCOVERY_PARTICIPANT_COMPACT	1419
5.207.2.62 DDS_SNIPPET_OPTIMIZATION_DISCOVERY_ENDPOINT_FAST	1420
5.207.2.63 DDS_SNIPPET_OPTIMIZATION_TRANSPORT_LARGE_BUFFERS	1420
5.207.2.64 DDS_SNIPPET_QOS_POLICY_RELIABILITY_RELIABLE	1420
5.207.2.65 DDS_SNIPPET_QOS_POLICY_RELIABILITY_BEST_EFFORT	1421
5.207.2.66 DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_LAST_1	1421
5.207.2.67 DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_ALL	1421
5.207.2.68 DDS_SNIPPET_QOS_POLICY_PUBLISH_MODE_ASYNCHRONOUS	1422
5.207.2.69 DDS_SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT_LOCAL	1422
5.207.2.70 DDS_SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT	1423
5.207.2.71 DDS_SNIPPET_QOS_POLICY_DURABILITY_PERSISTENT	1423
5.207.2.72 DDS_SNIPPET_QOS_POLICY_BATCHING_ENABLE	1424
5.207.2.73 DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_838MBPS	1424
5.207.2.74 DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_209MBPS	1424
5.207.2.75 DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_52MBPS	1425
5.207.2.76 DDS_SNIPPET_FEATURE_AUTO_TUNING_ENABLE	1425
5.207.2.77 DDS_SNIPPET_FEATURE_MONITORING_ENABLE	1426
5.207.2.78 DDS_SNIPPET_FEATURE_MONITORING2_ENABLE	1426

5.207.2.79 DDS_SNIPPET_FEATURE_SECURITY_ENABLE	1426
5.207.2.80 DDS_SNIPPET_FEATURE_TOPIC_QUERY_ENABLE	1427
5.207.2.81 DDS_SNIPPET_TRANSPORT_TCP_LAN_CLIENT	1427
5.207.2.82 DDS_SNIPPET_TRANSPORT_TCP_WAN_SYMMETRIC_CLIENT	1428
5.207.2.83 DDS_SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_SERVER	1428
5.207.2.84 DDS_SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_CLIENT	1428
5.207.2.85 DDS_SNIPPET_TRANSPORT_UDP_AVOID_IP_FRAGMENTATION	1429
5.207.2.86 DDS_SNIPPET_TRANSPORT_UDP_WAN	1429
5.207.2.87 DDS_SNIPPET_COMPATIBILITY_CONNEXT_MICRO_VERSION_2_4_3	1429
5.207.2.88 DDS_SNIPPET_COMPATIBILITY_OTHER_DDS_VENDORS_ENABLE	1430
5.207.2.89 DDS_SNIPPET_5_1_0_TRANSPORT_ENABLE	1430
5.208 DomainParticipantConfigParams	1430
5.208.1 Detailed Description	1431
5.208.2 Macro Definition Documentation	1431
5.208.2.1 DDS_DomainParticipantConfigParams_t_INITIALIZER	1431
5.208.3 Variable Documentation	1431
5.208.3.1 DDS_DOMAIN_ID_USE_XML_CONFIG	1431
5.208.3.2 DDS_ENTITY_NAME_USE_XML_CONFIG	1431
5.208.3.3 DDS_QOS_ELEMENT_NAME_USE_XML_CONFIG	1432
5.209 User-managed Threads	1432
5.209.1 Detailed Description	1432
5.209.2 Macro Definition Documentation	1433
5.209.2.1 DDS_ThreadFactory_INITIALIZER	1433
5.209.3 Typedef Documentation	1433
5.209.3.1 DDS_ThreadFactory_OnSpawnedFunction	1433
5.209.3.2 DDS_ThreadFactory_CreateThreadCallback	1433
5.209.3.3 DDS_ThreadFactory_DeleteThreadCallback	1434
5.210 Monitoring Library 2.0	1434
5.210.1 Detailed Description	1435
5.210.2 Function Documentation	1435
5.210.2.1 RTI_Monitoring_initialize()	1435
5.210.3 Variable Documentation	1436
5.210.3.1 RTI_MONITORING_PERIODIC_TOPIC_NAME	1436
5.210.3.2 RTI_MONITORING_EVENT_TOPIC_NAME	1436
5.210.3.3 RTI_MONITORING_LOGGING_TOPIC_NAME	1436
5.211 Version	1436
5.211.1 Detailed Description	1437
5.211.2 Function Documentation	1437
5.211.2.1 NDDS_Config_Version_get_product_version()	1437

5.211.2.2 NDDS_Config_Version_get_api_version()	1438
5.211.2.3 NDDS_Config_Version_get_core_version()	1438
5.211.2.4 NDDS_Config_Version_to_string()	1438
5.212 Logging	1438
5.212.1 Detailed Description	1442
5.212.2 Macro Definition Documentation	1442
5.212.2.1 NDDS_Config_LoggerDevice_INITIALIZER	1442
5.212.3 Typedef Documentation	1442
5.212.3.1 NDDS_Config_LoggerDeviceWriteFnc	1442
5.212.3.2 NDDS_Config_LoggerDeviceCloseFnc	1443
5.212.4 Enumeration Type Documentation	1443
5.212.4.1 NDDS_Config_LogVerbosity	1443
5.212.4.2 NDDS_Config_LogLevel	1444
5.212.4.3 NDDS_Config_SyslogLevel	1444
5.212.4.4 NDDS_Config_LogCategory	1445
5.212.4.5 NDDS_Config_LogPrintFormat	1446
5.212.4.6 NDDS_Config_LogFacility	1447
5.212.4.7 NDDS_Config_SyslogVerbosity	1447
5.212.5 Function Documentation	1448
5.212.5.1 NDDS_Config_Logger_get_instance()	1448
5.212.5.2 NDDS_Config_Logger_get_instancel()	1449
5.212.5.3 NDDS_Config_Logger_get_verbosity()	1449
5.212.5.4 NDDS_Config_Logger_get_verbosity_by_category()	1450
5.212.5.5 NDDS_Config_Logger_set_verbosity()	1450
5.212.5.6 NDDS_Config_Logger_set_verbosity_by_category()	1450
5.212.5.7 NDDS_Config_Logger_get_output_file()	1450
5.212.5.8 NDDS_Config_Logger_set_output_file()	1451
5.212.5.9 NDDS_Config_Logger_set_output_file_name()	1451
5.212.5.10 NDDS_Config_Logger_set_output_file_set()	1451
5.212.5.11 NDDS_Config_Logger_get_print_format()	1452
5.212.5.12 NDDS_Config_Logger_get_print_format_by_log_level()	1452
5.212.5.13 NDDS_Config_Logger_set_print_format()	1452
5.212.5.14 NDDS_Config_Logger_set_print_format_by_log_level()	1452
5.212.5.15 NDDS_Config_Logger_emergency()	1453
5.212.5.16 NDDS_Config_Logger_alert()	1453
5.212.5.17 NDDS_Config_Logger_critical()	1454
5.212.5.18 NDDS_Config_Logger_error()	1454
5.212.5.19 NDDS_Config_Logger_warning()	1455
5.212.5.20 NDDS_Config_Logger_notice()	1455

5.212.5.21 NDDS_Config_Logger_informational()	1456
5.212.5.22 NDDS_Config_Logger_debug()	1456
5.212.5.23 NDDS_Config_Logger_get_output_device()	1457
5.212.5.24 NDDS_Config_Logger_set_output_device()	1457
5.213 Activity Context	1458
5.213.1 Detailed Description	1459
5.213.2 Macro Definition Documentation	1460
5.213.2.1 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_DEFAULT	1460
5.213.2.2 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_NONE	1460
5.213.2.3 NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_ALL	1460
5.213.3 Typedef Documentation	1460
5.213.3.1 NDDS_Config_ActivityContextAttributeKindMask	1460
5.213.4 Enumeration Type Documentation	1460
5.213.4.1 NDDS_Config_ActivityContextAttributeKind	1460
5.213.5 Function Documentation	1462
5.213.5.1 NDDS_Config_ActivityContext_set_attribute_mask()	1463
5.214 Compliance Configuration	1463
5.214.1 Detailed Description	1464
5.214.2 Macro Definition Documentation	1464
5.214.2.1 NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_VENDOR	1464
5.214.2.2 NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_DEFAULT	1465
5.214.2.3 NDDS_CONFIG_RPC_COMPLIANCE_MASK_VENDOR	1465
5.214.2.4 NDDS_CONFIG_RPC_COMPLIANCE_MASK_DEFAULT	1465
5.214.3 Typedef Documentation	1465
5.214.3.1 NDDS_Config_XTypesComplianceMask	1466
5.214.3.2 NDDS_Config_RpcComplianceMask	1466
5.214.4 Enumeration Type Documentation	1467
5.214.4.1 NDDS_Config_XTypesComplianceMaskBits	1467
5.214.4.2 NDDS_Config_RpcComplianceMaskBits	1473
5.214.5 Function Documentation	1474
5.214.5.1 NDDS_Config_get_xtypes_compliance_mask()	1474
5.214.5.2 NDDS_Config_set_xtypes_compliance_mask()	1475
5.214.5.3 NDDS_Config_load_compliance_masks()	1475
5.214.5.4 NDDS_Config_get_rpc_compliance_mask()	1476
5.214.5.5 NDDS_Config_set_rpc_compliance_mask()	1476
5.215 Heap Monitoring	1477
5.215.1 Detailed Description	1478
5.215.2 Macro Definition Documentation	1478
5.215.2.1 NDDS_Utility_HeapMonitoringParams_INITIALIZER	1478

5.215.3 Typedef Documentation	1478
5.215.3.1 NDDS_Utility_HeapMonitoringParams_t	1478
5.215.4 Enumeration Type Documentation	1478
5.215.4.1 NDDS_Utility_HeapMonitoringSnapshotOutputFormat	1478
5.215.4.2 NDDS_Utility_HeapMonitoringSnapshotContentFormat	1479
5.215.5 Function Documentation	1479
5.215.5.1 NDDS_Utility_enable_heap_monitoring()	1479
5.215.5.2 NDDS_Utility_enable_heap_monitoring_w_params()	1480
5.215.5.3 NDDS_Utility_disable_heap_monitoring()	1480
5.215.5.4 NDDS_Utility_pause_heap_monitoring()	1481
5.215.5.5 NDDS_Utility_resume_heap_monitoring()	1481
5.215.5.6 NDDS_Utility_take_heap_snapshot()	1482
5.216 Network Capture	1483
5.216.1 Detailed Description	1485
5.216.2 Capturing	1486
5.216.3 Macro Definition Documentation	1486
5.216.3.1 NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_DEFAULT	1486
5.216.3.2 NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_NONE	1487
5.216.3.3 NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_ALL	1487
5.216.3.4 NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_DEFAULT	1487
5.216.3.5 NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_NONE	1487
5.216.3.6 NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_ALL	1487
5.216.4 Typedef Documentation	1488
5.216.4.1 NDDS_Utility_NetworkCaptureContentKindMask	1488
5.216.4.2 NDDS_Utility_NetworkCaptureTrafficKindMask	1488
5.216.4.3 NDDS_Utility_NetworkCaptureParams_t	1488
5.216.5 Enumeration Type Documentation	1488
5.216.5.1 NDDS_Utility_NetworkCaptureContentKind	1488
5.216.5.2 NDDS_Utility_NetworkCaptureTrafficKind	1489
5.216.6 Function Documentation	1489
5.216.6.1 NDDS_Utility_enable_network_capture()	1489
5.216.6.2 NDDS_Utility_disable_network_capture()	1490
5.216.6.3 NDDS_Utility_set_default_network_capture_params()	1491
5.216.6.4 NDDS_Utility_start_network_capture()	1491
5.216.6.5 NDDS_Utility_start_network_capture_for_participant()	1492
5.216.6.6 NDDS_Utility_start_network_capture_w_params()	1493
5.216.6.7 NDDS_Utility_start_network_capture_w_params_for_participant()	1493
5.216.6.8 NDDS_Utility_stop_network_capture()	1494
5.216.6.9 NDDS_Utility_stop_network_capture_for_participant()	1495

5.216.6.10 NDDS_Utility_pause_network_capture()	1495
5.216.6.11 NDDS_Utility_pause_network_capture_for_participant()	1496
5.216.6.12 NDDS_Utility_resume_network_capture()	1496
5.216.6.13 NDDS_Utility_resume_network_capture_for_participant()	1497
5.216.7 Variable Documentation	1497
5.216.7.1 NDDS_UTILITY_NETWORK_CAPTURE_PARAMETERS_DEFAULT	1497
5.217 Other Utilities	1498
5.217.1 Detailed Description	1498
5.217.2 Function Documentation	1498
5.217.2.1 NDDS_LM_unload_activation_library()	1498
5.217.2.2 NDDS_Config_enable_automatic_tss_cleanup()	1499
5.217.2.3 NDDS_Utility_sleep()	1499
5.217.2.4 NDDS_Utility_spin()	1499
5.217.2.5 NDDS_Utility_get_spin_per_microsecond()	1500
5.218 Octet Buffer Support	1500
5.218.1 Detailed Description	1501
5.218.2 Conventions	1501
5.218.3 Usage	1501
5.218.4 Function Documentation	1502
5.218.4.1 DDS_OctetBuffer_alloc()	1502
5.218.4.2 DDS_OctetBuffer_dup()	1502
5.218.4.3 DDS_OctetBuffer_free()	1503
5.219 SampleProcessor	1503
5.219.1 Detailed Description	1504
5.219.2 Macro Definition Documentation	1504
5.219.2.1 DDS_SampleHandler_INITIALIZER	1504
5.219.3 Typedef Documentation	1505
5.219.3.1 DDS_SampleProcessor	1505
5.219.4 SampleProcessor Thread Safety	1506
5.219.5 Function Documentation	1506
5.219.5.1 DDS_SampleProcessor_attach_reader()	1507
5.219.5.2 DDS_SampleProcessor_detach_reader()	1507
5.219.5.3 DDS_SampleProcessor_lookup_sample_handler()	1508
5.219.5.4 DDS_SampleProcessor_get_datareaders()	1509
5.219.5.5 DDS_SampleProcessor_delete()	1509
5.219.5.6 DDS_SampleProcessor_new()	1510
5.219.5.7 DDS_SampleProcessor_new_with_aws()	1510
5.220 Sequence Support	1511
5.220.1 Detailed Description	1513

5.220.2 Macro Definition Documentation	1513
5.220.2.1 DDS_SEQUENCE_INITIALIZER	1513
5.220.3 Function Documentation	1513
5.220.3.1 FooSeq_initialize()	1513
5.220.3.2 FooSeq_get_maximum()	1514
5.220.3.3 FooSeq_set_maximum()	1514
5.220.3.4 FooSeq_get_length()	1515
5.220.3.5 FooSeq_set_length()	1515
5.220.3.6 FooSeq_ensure_length()	1516
5.220.3.7 FooSeq_get()	1517
5.220.3.8 FooSeq_get_reference()	1517
5.220.3.9 FooSeq_copy_no_alloc()	1518
5.220.3.10 FooSeq_copy()	1519
5.220.3.11 FooSeq_from_array()	1519
5.220.3.12 FooSeq_to_array()	1520
5.220.3.13 FooSeq_loan_contiguous()	1521
5.220.3.14 FooSeq_loan_discontiguous()	1522
5.220.3.15 FooSeq_unloan()	1523
5.220.3.16 FooSeq_get_contiguous_buffer()	1523
5.220.3.17 FooSeq_get_discontiguous_buffer()	1524
5.220.3.18 FooSeq_has_ownership()	1525
5.220.3.19 FooSeq_finalize()	1525
5.221 String Support	1526
5.221.1 Detailed Description	1526
5.221.2 String Conventions	1527
5.221.3 Usage	1527
5.221.4 Function Documentation	1528
5.221.4.1 DDS_String_alloc()	1528
5.221.4.2 DDS_String_dup()	1528
5.221.4.3 DDS_String_replace()	1529
5.221.4.4 DDS_String_free()	1529
5.221.4.5 DDS_Wstring_alloc()	1530
5.221.4.6 DDS_Wstring_length()	1530
5.221.4.7 DDS_Wstring_copy()	1531
5.221.4.8 DDS_Wstring_copy_and_widen()	1531
5.221.4.9 DDS_Wstring_dup()	1532
5.221.4.10 DDS_Wstring_dup_and_widen()	1532
5.221.4.11 DDS_Wstring_free()	1533

6 Data Structure Documentation	1535
6.1 DDS_AcknowledgmentInfo Struct Reference	1535
6.1.1 Detailed Description	1535
6.1.2 Field Documentation	1535
6.1.2.1 subscription_handle	1536
6.1.2.2 sample_identity	1536
6.1.2.3 valid_response_data	1536
6.1.2.4 response_data	1536
6.2 DDS_AckResponseData_t Struct Reference	1536
6.2.1 Detailed Description	1537
6.2.2 Field Documentation	1537
6.2.2.1 value	1537
6.3 DDS_AllocationSettings_t Struct Reference	1537
6.3.1 Detailed Description	1537
6.3.2 Field Documentation	1538
6.3.2.1 initial_count	1538
6.3.2.2 max_count	1538
6.3.2.3 incremental_count	1538
6.4 DDS_AnnotationParameterValue Struct Reference	1538
6.4.1 Detailed Description	1539
6.5 DDS_AsynchronousPublisherQosPolicy Struct Reference	1539
6.5.1 Detailed Description	1539
6.5.2 Usage	1540
6.5.3 Field Documentation	1540
6.5.3.1 disable_asynchronous_write	1540
6.5.3.2 thread	1541
6.5.3.3 disable_asynchronous_batch	1541
6.5.3.4 asynchronous_batch_thread	1542
6.5.3.5 disable_topic_query_publication	1542
6.5.3.6 topic_query_publication_thread	1543
6.6 DDS_AsyncWaitSetListener Struct Reference	1543
6.6.1 Detailed Description	1544
6.6.2 Field Documentation	1544
6.6.2.1 listener_data	1544
6.6.2.2 on_thread_spawned	1544
6.6.2.3 on_thread_deleted	1544
6.6.2.4 on_wait_timeout	1544
6.7 DDS_AsyncWaitSetProperty_t Struct Reference	1544
6.7.1 Detailed Description	1545

6.7.2 Field Documentation	1545
6.7.2.1 waitset_property	1545
6.7.2.2 thread_pool_size	1546
6.7.2.3 thread_settings	1546
6.7.2.4 thread_name_prefix	1546
6.7.2.5 wait_timeout	1547
6.7.2.6 level	1547
6.8 DDS_AvailabilityQosPolicy Struct Reference	1547
6.8.1 Detailed Description	1548
6.8.2 Usage	1548
6.8.3 Consistency	1549
6.8.4 Field Documentation	1550
6.8.4.1 enable_required_subscriptions	1550
6.8.4.2 max_data_availability_waiting_time	1550
6.8.4.3 max_endpoint_availability_waiting_time	1551
6.8.4.4 required_matched_endpoint_groups	1551
6.9 DDS_BatchQosPolicy Struct Reference	1552
6.9.1 Detailed Description	1552
6.9.2 Field Documentation	1552
6.9.2.1 enable	1553
6.9.2.2 max_data_bytes	1553
6.9.3 Consistency	1553
6.9.3.1 max_samples	1554
6.9.4 Consistency	1554
6.9.4.1 max_flush_delay	1554
6.9.5 Consistency	1554
6.9.5.1 source_timestamp_resolution	1555
6.9.6 Consistency	1555
6.9.6.1 thread_safe_write	1555
6.9.7 Consistency	1556
6.10 DDS_BooleanSeq Struct Reference	1556
6.10.1 Detailed Description	1556
6.11 DDS_BuiltinTopicKey_t Struct Reference	1556
6.11.1 Detailed Description	1557
6.12 DDS_BuiltinTopicReaderResourceLimits_t Struct Reference	1557
6.12.1 Detailed Description	1558
6.12.2 Field Documentation	1558
6.12.2.1 initial_samples	1558
6.12.2.2 max_samples	1559

6.12.2.3 initial_infos	1559
6.12.2.4 max_infos	1559
6.12.2.5 initial_outstanding_reads	1560
6.12.2.6 max_outstanding_reads	1560
6.12.2.7 max_samples_per_read	1560
6.12.2.8 disable_fragmentation_support	1561
6.12.2.9 max_fragmented_samples	1561
6.12.2.10 initial_fragmented_samples	1562
6.12.2.11 max_fragmented_samples_per_remote_writer	1562
6.12.2.12 max_fragments_per_sample	1563
6.12.2.13 dynamically_allocate_fragmented_samples	1563
6.13 DDS_ChannelSettings_t Struct Reference	1564
6.13.1 Detailed Description	1564
6.13.2 Field Documentation	1564
6.13.2.1 multicast_settings	1564
6.13.2.2 filter_expression	1565
6.13.2.3 priority	1565
6.14 DDS_ChannelSettingsSeq Struct Reference	1566
6.14.1 Detailed Description	1566
6.15 DDS_CharSeq Struct Reference	1566
6.15.1 Detailed Description	1566
6.16 DDS_CoherentSetInfo_t Struct Reference	1566
6.16.1 Detailed Description	1567
6.16.2 Field Documentation	1567
6.16.2.1 group_guid	1567
6.16.2.2 coherent_set_sequence_number	1567
6.16.2.3 group_coherent_set_sequence_number	1568
6.16.2.4 incomplete_coherent_set	1568
6.17 DDS_CompressionSettings_t Struct Reference	1568
6.17.1 Detailed Description	1568
6.17.2 Field Documentation	1568
6.17.2.1 compression_ids	1569
6.17.2.2 writer_compression_level	1569
6.17.2.3 writer_compression_threshold	1570
6.18 DDS_ConditionHandler Struct Reference	1570
6.18.1 Detailed Description	1570
6.18.2 Field Documentation	1571
6.18.2.1 handler_data	1571
6.18.2.2 on_condition_triggered	1571

6.19 DDS_ConditionSeq Struct Reference	1571
6.19.1 Detailed Description	1571
6.20 DDS_ContentFilter Struct Reference	1571
6.20.1 Detailed Description	1572
6.20.2 Field Documentation	1572
6.20.2.1 compile	1573
6.20.2.2 writer_compile	1574
6.20.2.3 evaluate	1574
6.20.2.4 writer_evaluate	1576
6.20.2.5 finalize	1576
6.20.2.6 writer_finalize	1577
6.20.2.7 writer_attach	1577
6.20.2.8 writer_detach	1578
6.20.2.9 writer_return_loan	1578
6.20.2.10 filter_data	1578
6.21 DDS_ContentFilterProperty_t Struct Reference	1579
6.21.1 Detailed Description	1579
6.21.2 Field Documentation	1579
6.21.2.1 content_filter_topic_name	1579
6.21.2.2 related_topic_name	1579
6.21.2.3 filter_class_name	1580
6.21.2.4 filter_expression	1580
6.21.2.5 expression_parameters	1580
6.22 DDS_Cookie_t Struct Reference	1580
6.22.1 Detailed Description	1580
6.22.2 Field Documentation	1580
6.22.2.1 value	1581
6.23 DDS_CookieSeq Struct Reference	1581
6.23.1 Detailed Description	1581
6.24 DDS_DatabaseQosPolicy Struct Reference	1581
6.24.1 Detailed Description	1582
6.24.2 Field Documentation	1582
6.24.2.1 thread	1582
6.24.2.2 shutdown_timeout	1583
6.24.2.3 cleanup_period	1583
6.24.2.4 shutdown_cleanup_period	1584
6.24.2.5 initial_records	1584
6.24.2.6 max_skiplist_level	1584
6.24.2.7 max_weak_references	1585

6.24.2.8 initial_weak_references	1585
6.25 DDS_DataReaderCacheStatus Struct Reference	1586
6.25.1 Detailed Description	1587
6.25.2 Field Documentation	1587
6.25.2.1 sample_count_peak	1587
6.25.2.2 sample_count	1587
6.25.2.3 old_source_timestamp_dropped_sample_count	1588
6.25.2.4 tolerance_source_timestamp_dropped_sample_count	1588
6.25.2.5 ownership_dropped_sample_count	1588
6.25.2.6 content_filter_dropped_sample_count	1588
6.25.2.7 time_based_filter_dropped_sample_count	1589
6.25.2.8 expired_dropped_sample_count	1589
6.25.2.9 virtual_duplicate_dropped_sample_count	1589
6.25.2.10 replaced_dropped_sample_count	1590
6.25.2.11 writer_removed_batch_sample_dropped_sample_count	1590
6.25.2.12 total_samples_dropped_by_instance_replacement	1590
6.25.2.13 alive_instance_count	1590
6.25.2.14 alive_instance_count_peak	1591
6.25.2.15 no_writers_instance_count	1591
6.25.2.16 no_writers_instance_count_peak	1591
6.25.2.17 disposed_instance_count	1591
6.25.2.18 disposed_instance_count_peak	1592
6.25.2.19 detached_instance_count	1592
6.25.2.20 detached_instance_count_peak	1592
6.25.2.21 compressed_sample_count	1592
6.26 DDS_DataReaderListener Struct Reference	1593
6.26.1 Detailed Description	1593
6.26.2 Field Documentation	1594
6.26.2.1 as_listener	1594
6.26.2.2 on_requested_deadline_missed	1594
6.26.2.3 on_requested_incompatible_qos	1594
6.26.2.4 on_sample_rejected	1594
6.26.2.5 on_liveliness_changed	1595
6.26.2.6 on_data_available	1595
6.26.2.7 on_subscription_matched	1595
6.26.2.8 on_sample_lost	1595
6.27 DDS_DataReaderProtocolQosPolicy Struct Reference	1596
6.27.1 Detailed Description	1596
6.27.2 Field Documentation	1597

6.27.2.1 virtual_guid	1597
6.27.2.2 rtps_object_id	1597
6.27.2.3 expects_inline_qos	1598
6.27.2.4 disable_positive_acks	1598
6.27.2.5 propagate_dispose_of_unregistered_instances	1599
6.27.2.6 propagate_unregister_of_disposed_instances	1599
6.27.2.7 rtps_reliable_reader	1600
6.28 DDS_DataReaderProtocolStatus Struct Reference	1600
6.28.1 Detailed Description	1602
6.28.2 Field Documentation	1602
6.28.2.1 received_sample_count	1602
6.28.2.2 received_sample_count_change	1603
6.28.2.3 received_sample_bytes	1603
6.28.2.4 received_sample_bytes_change	1604
6.28.2.5 duplicate_sample_count	1604
6.28.2.6 duplicate_sample_count_change	1604
6.28.2.7 duplicate_sample_bytes	1604
6.28.2.8 duplicate_sample_bytes_change	1604
6.28.2.9 filtered_sample_count	1605
6.28.2.10 filtered_sample_count_change	1605
6.28.2.11 filtered_sample_bytes	1605
6.28.2.12 filtered_sample_bytes_change	1605
6.28.2.13 received_heartbeat_count	1605
6.28.2.14 received_heartbeat_count_change	1606
6.28.2.15 received_heartbeat_bytes	1606
6.28.2.16 received_heartbeat_bytes_change	1606
6.28.2.17 sent_ack_count	1606
6.28.2.18 sent_ack_count_change	1606
6.28.2.19 sent_ack_bytes	1607
6.28.2.20 sent_ack_bytes_change	1607
6.28.2.21 sent_nack_count	1607
6.28.2.22 sent_nack_count_change	1607
6.28.2.23 sent_nack_bytes	1607
6.28.2.24 sent_nack_bytes_change	1607
6.28.2.25 received_gap_count	1608
6.28.2.26 received_gap_count_change	1608
6.28.2.27 received_gap_bytes	1608
6.28.2.28 received_gap_bytes_change	1608
6.28.2.29 rejected_sample_count	1608

6.28.2.30 rejected_sample_count_change	1609
6.28.2.31 first_available_sample_sequence_number	1609
6.28.2.32 last_available_sample_sequence_number	1609
6.28.2.33 last_committed_sample_sequence_number	1609
6.28.2.34 uncommitted_sample_count	1609
6.28.2.35 out_of_range_rejected_sample_count	1610
6.28.2.36 received_fragment_count	1610
6.28.2.37 dropped_fragment_count	1610
6.28.2.38 reassembled_sample_count	1610
6.28.2.39 sent_nack_fragment_count	1610
6.28.2.40 sent_nack_fragment_bytes	1611
6.29 DDS_DataReaderQos Struct Reference	1611
6.29.1 Detailed Description	1612
6.29.2 Field Documentation	1613
6.29.2.1 durability	1613
6.29.2.2 deadline	1613
6.29.2.3 latency_budget	1613
6.29.2.4 liveliness	1614
6.29.2.5 reliability	1614
6.29.2.6 destination_order	1614
6.29.2.7 history	1614
6.29.2.8 resource_limits	1614
6.29.2.9 user_data	1614
6.29.2.10 ownership	1615
6.29.2.11 time_based_filter	1615
6.29.2.12 reader_data_lifecycle	1615
6.29.2.13 representation	1615
6.29.2.14 type_consistency	1615
6.29.2.15 data_tags	1615
6.29.2.16 reader_resource_limits	1616
6.29.2.17 protocol	1616
6.29.2.18 transport_selection	1616
6.29.2.19 unicast	1616
6.29.2.20 multicast	1616
6.29.2.21 property	1617
6.29.2.22 service	1617
6.29.2.23 availability	1617
6.29.2.24 subscription_name	1617
6.29.2.25 transport_priority	1617

6.29.2.26 type_support	1617
6.30 DDS_DataReaderResourceLimitsInstanceReplacementSettings Struct Reference	1618
6.30.1 Detailed Description	1618
6.30.2 Field Documentation	1618
6.30.2.1 alive_instance_removal	1618
6.30.2.2 disposed_instance_removal	1619
6.30.2.3 no_writers_instance_removal	1619
6.31 DDS_DataReaderResourceLimitsQosPolicy Struct Reference	1619
6.31.1 Detailed Description	1621
6.31.2 Field Documentation	1621
6.31.2.1 max_remote_writers	1622
6.31.2.2 max_remote_writers_per_instance	1622
6.31.2.3 max_samples_per_remote_writer	1623
6.31.2.4 max_infos	1623
6.31.2.5 initial_remote_writers	1624
6.31.2.6 initial_remote_writers_per_instance	1624
6.31.2.7 initial_infos	1625
6.31.2.8 initial_outstanding_reads	1625
6.31.2.9 max_outstanding_reads	1625
6.31.2.10 max_samples_per_read	1626
6.31.2.11 disable_fragmentation_support	1626
6.31.2.12 max_fragmented_samples	1627
6.31.2.13 initial_fragmented_samples	1627
6.31.2.14 max_fragmented_samples_per_remote_writer	1628
6.31.2.15 max_fragments_per_sample	1628
6.31.2.16 dynamically_allocate_fragmented_samples	1629
6.31.2.17 max_total_instances	1629
6.31.2.18 max_remote_virtual_writers	1630
6.31.2.19 initial_remote_virtual_writers	1631
6.31.2.20 max_remote_virtual_writers_per_instance	1631
6.31.2.21 initial_remote_virtual_writers_per_instance	1632
6.31.2.22 max_remote_writers_per_sample	1632
6.31.2.23 max_query_condition_filters	1633
6.31.2.24 max_app_ack_response_length	1633
6.31.2.25 keep_minimum_state_for_instances	1634
6.31.2.26 initial_topic_queries	1634
6.31.2.27 max_topic_queries	1635
6.31.2.28 shmem_ref_transfer_mode_attached_segment_allocation	1635
6.31.2.29 instance_replacement	1636

6.31.2.30 autopurge_remote_not_alive_writer_delay	1637
6.32 DDS_DataReaderSeq Struct Reference	1637
6.32.1 Detailed Description	1638
6.33 DDS_DataRepresentationIdSeq Struct Reference	1638
6.33.1 Detailed Description	1638
6.34 DDS_DataRepresentationQosPolicy Struct Reference	1638
6.34.1 Detailed Description	1639
6.34.2 Field Documentation	1640
6.34.2.1 value	1640
6.34.2.2 compression_settings	1640
6.35 DDS_DataTags Struct Reference	1641
6.35.1 Detailed Description	1641
6.35.2 Field Documentation	1641
6.35.2.1 tags	1641
6.36 DDS_DataWriterCacheStatus Struct Reference	1641
6.36.1 Detailed Description	1642
6.36.2 Field Documentation	1642
6.36.2.1 sample_count_peak	1642
6.36.2.2 sample_count	1643
6.36.2.3 alive_instance_count	1643
6.36.2.4 alive_instance_count_peak	1643
6.36.2.5 disposed_instance_count	1643
6.36.2.6 disposed_instance_count_peak	1643
6.36.2.7 unregistered_instance_count	1644
6.36.2.8 unregistered_instance_count_peak	1644
6.37 DDS_DataWriterListener Struct Reference	1644
6.37.1 Detailed Description	1645
6.37.2 Field Documentation	1645
6.37.2.1 as_listener	1645
6.37.2.2 on_offered_deadline_missed	1645
6.37.2.3 on_offered_incompatible_qos	1646
6.37.2.4 on_liveliness_lost	1646
6.37.2.5 on_publication_matched	1646
6.37.2.6 on_reliable_writer_cache_changed	1646
6.37.2.7 on_reliable_reader_activity_changed	1647
6.37.2.8 on_sample_removed	1647
6.37.2.9 on_instance_replaced	1647
6.37.2.10 on_application_acknowledgment	1648
6.37.2.11 on_service_request_accepted	1648

6.37.2.12 on_empty_virtual_writer	1648
6.38 DDS_DataWriterProtocolQosPolicy Struct Reference	1649
6.38.1 Detailed Description	1649
6.38.2 Field Documentation	1650
6.38.2.1 virtual_guid	1650
6.38.2.2 rtps_object_id	1650
6.38.2.3 push_on_write	1651
6.38.2.4 disable_positive_acks	1651
6.38.2.5 disable_inline_keyhash	1652
6.38.2.6 serialize_key_with_dispose	1652
6.38.2.7 propagate_app_ack_with_no_response	1653
6.38.2.8 rtps_reliable_writer	1653
6.38.2.9 initial_virtual_sequence_number	1654
6.39 DDS_DataWriterProtocolStatus Struct Reference	1654
6.39.1 Detailed Description	1656
6.39.2 Field Documentation	1657
6.39.2.1 pushed_sample_count	1657
6.39.2.2 pushed_sample_count_change	1657
6.39.2.3 pushed_sample_bytes	1657
6.39.2.4 pushed_sample_bytes_change	1657
6.39.2.5 filtered_sample_count	1658
6.39.2.6 filtered_sample_count_change	1658
6.39.2.7 filtered_sample_bytes	1658
6.39.2.8 filtered_sample_bytes_change	1658
6.39.2.9 sent_heartbeat_count	1658
6.39.2.10 sent_heartbeat_count_change	1659
6.39.2.11 sent_heartbeat_bytes	1659
6.39.2.12 sent_heartbeat_bytes_change	1659
6.39.2.13 pulled_sample_count	1659
6.39.2.14 pulled_sample_count_change	1659
6.39.2.15 pulled_sample_bytes	1660
6.39.2.16 pulled_sample_bytes_change	1660
6.39.2.17 received_ack_count	1660
6.39.2.18 received_ack_count_change	1660
6.39.2.19 received_ack_bytes	1660
6.39.2.20 received_ack_bytes_change	1661
6.39.2.21 received_nack_count	1661
6.39.2.22 received_nack_count_change	1661
6.39.2.23 received_nack_bytes	1661

6.39.2.24 received_nack_bytes_change	1661
6.39.2.25 sent_gap_count	1662
6.39.2.26 sent_gap_count_change	1662
6.39.2.27 sent_gap_bytes	1662
6.39.2.28 sent_gap_bytes_change	1662
6.39.2.29 rejected_sample_count	1662
6.39.2.30 rejected_sample_count_change	1662
6.39.2.31 send_window_size	1663
6.39.2.32 first_available_sample_sequence_number	1663
6.39.2.33 last_available_sample_sequence_number	1663
6.39.2.34 first_unacknowledged_sample_sequence_number	1663
6.39.2.35 first_available_sample_virtual_sequence_number	1663
6.39.2.36 last_available_sample_virtual_sequence_number	1664
6.39.2.37 first_unacknowledged_sample_virtual_sequence_number	1664
6.39.2.38 first_unacknowledged_sample_subscription_handle	1664
6.39.2.39 first_unelapsed_keep_duration_sample_sequence_number	1664
6.39.2.40 pushed_fragment_count	1664
6.39.2.41 pushed_fragment_bytes	1665
6.39.2.42 pulled_fragment_count	1665
6.39.2.43 pulled_fragment_bytes	1665
6.39.2.44 received_nack_fragment_count	1665
6.39.2.45 received_nack_fragment_bytes	1665
6.40 DDS_DataWriterQos Struct Reference	1666
6.40.1 Detailed Description	1667
6.40.2 Field Documentation	1668
6.40.2.1 durability	1668
6.40.2.2 durability_service	1668
6.40.2.3 deadline	1668
6.40.2.4 latency_budget	1668
6.40.2.5 liveliness	1669
6.40.2.6 reliability	1669
6.40.2.7 destination_order	1669
6.40.2.8 history	1669
6.40.2.9 resource_limits	1669
6.40.2.10 transport_priority	1669
6.40.2.11 lifespan	1670
6.40.2.12 user_data	1670
6.40.2.13 ownership	1670
6.40.2.14 ownership_strength	1670

6.40.2.15 writer_data_lifecycle	1670
6.40.2.16 representation	1670
6.40.2.17 data_tags	1671
6.40.2.18 writer_resource_limits	1671
6.40.2.19 protocol	1671
6.40.2.20 transport_selection	1671
6.40.2.21 unicast	1671
6.40.2.22 publish_mode	1672
6.40.2.23 property	1672
6.40.2.24 service	1672
6.40.2.25 batch	1672
6.40.2.26 multi_channel	1672
6.40.2.27 availability	1672
6.40.2.28 publication_name	1673
6.40.2.29 topic_query_dispatch	1673
6.40.2.30 transfer_mode	1673
6.40.2.31 type_support	1673
6.41 DDS_DataWriterResourceLimitsQosPolicy Struct Reference	1673
6.41.1 Detailed Description	1674
6.41.2 Field Documentation	1675
6.41.2.1 initial_concurrent_blocking_threads	1675
6.41.2.2 max_concurrent_blocking_threads	1676
6.41.2.3 max_remote_reader_filters	1676
6.41.2.4 initial_batches	1677
6.41.2.5 max_batches	1677
6.41.2.6 instance_replacement	1678
6.41.2.7 replace_empty_instances	1678
6.41.2.8 autoregister_instances	1679
6.41.2.9 initial_virtual_writers	1679
6.41.2.10 max_virtual_writers	1679
6.41.2.11 max_remote_readers	1680
6.41.2.12 max_app_ack_remote_readers	1680
6.41.2.13 initial_active_topic_queries	1680
6.41.2.14 max_active_topic_queries	1681
6.41.2.15 writer_loaned_sample_allocation	1681
6.41.2.16 initialize_writer_loaned_sample	1682
6.42 DDS_DataWriterShmemRefTransferModeSettings Struct Reference	1682
6.42.1 Detailed Description	1682
6.42.2 Field Documentation	1682

6.42.2.1 enable_data_consistency_check	1683
6.43 DDS_DataWriterTransferModeQosPolicy Struct Reference	1683
6.43.1 Detailed Description	1683
6.43.2 Field Documentation	1683
6.43.2.1 shmem_ref_settings	1684
6.44 DDS_DeadlineQosPolicy Struct Reference	1684
6.44.1 Detailed Description	1684
6.44.2 Usage	1685
6.44.3 Compatibility	1685
6.44.4 Consistency	1685
6.44.5 Field Documentation	1685
6.44.5.1 period	1686
6.45 DDS_DestinationOrderQosPolicy Struct Reference	1686
6.45.1 Detailed Description	1686
6.45.2 Usage	1687
6.45.3 Compatibility	1688
6.45.4 Field Documentation	1688
6.45.4.1 kind	1688
6.45.4.2 scope	1688
6.45.4.3 source_timestamp_tolerance	1689
6.46 DDS_DiscoveryConfigQosPolicy Struct Reference	1689
6.46.1 Detailed Description	1692
6.46.2 Field Documentation	1692
6.46.2.1 participant_liveliness_lease_duration	1692
6.46.2.2 participant_liveliness_assert_period	1693
6.46.2.3 participant_announcement_period	1693
6.46.2.4 remote_participant_purge_kind	1694
6.46.2.5 max_liveliness_loss_detection_period	1694
6.46.2.6 initial_participant_announcements	1695
6.46.2.7 new_remote_participant_announcements	1695
6.46.2.8 min_initial_participant_announcement_period	1695
6.46.2.9 max_initial_participant_announcement_period	1696
6.46.2.10 participant_reader_resource_limits	1696
6.46.2.11 publication_reader	1697
6.46.2.12 publication_reader_resource_limits	1697
6.46.2.13 subscription_reader	1698
6.46.2.14 subscription_reader_resource_limits	1698
6.46.2.15 publication_writer	1699
6.46.2.16 publication_writer_data_lifecycle	1699

6.46.2.17 subscription_writer	1700
6.46.2.18 subscription_writer_data_lifecycle	1701
6.46.2.19 builtin_discovery_plugins	1701
6.46.2.20 enabled_builtin_channels	1701
6.46.2.21 participant_message_reader_reliability_kind	1702
6.46.2.22 participant_message_reader	1702
6.46.2.23 participant_message_writer	1703
6.46.2.24 publication_writer_publish_mode	1703
6.46.2.25 subscription_writer_publish_mode	1704
6.46.2.26 asynchronous_publisher	1704
6.46.2.27 default_domain_announcement_period	1705
6.46.2.28 ignore_default_domain_announcements	1705
6.46.2.29 service_request_writer	1706
6.46.2.30 service_request_writer_data_lifecycle	1706
6.46.2.31 service_request_writer_publish_mode	1707
6.46.2.32 service_request_reader	1707
6.46.2.33 locator_reachability_assert_period	1708
6.46.2.34 locator_reachability_lease_duration	1708
6.46.2.35 locator_reachability_change_detection_period	1709
6.46.2.36 secure_volatile_writer	1709
6.46.2.37 secure_volatile_writer_publish_mode	1710
6.46.2.38 secure_volatile_reader	1710
6.46.2.39 endpoint_type_object_lb_serialization_threshold	1711
6.46.2.40 dns_tracker_polling_period	1711
6.46.2.41 participant_configuration_writer_publish_mode	1712
6.46.2.42 participant_configuration_writer	1713
6.46.2.43 participant_configuration_writer_data_lifecycle	1713
6.46.2.44 participant_configuration_reader	1714
6.46.2.45 participant_configuration_reader_resource_limits	1714
6.47 DDS_DiscoveryQosPolicy Struct Reference	1715
6.47.1 Detailed Description	1715
6.47.2 Usage	1715
6.47.3 Field Documentation	1716
6.47.3.1 enabled_transports	1716
6.47.3.2 initial_peers	1716
6.47.3.3 multicast_receive_addresses	1717
6.47.3.4 metatraffic_transport_priority	1718
6.47.3.5 accept_unknown_peers	1718
6.47.3.6 enable_endpoint_discovery	1719

6.48 DDS_DomainParticipantConfigParams_t Struct Reference	1719
6.48.1 Detailed Description	1719
6.48.2 Field Documentation	1720
6.48.2.1 domain_id	1720
6.48.2.2 participant_name	1720
6.48.2.3 participant_qos_library_name	1720
6.48.2.4 participant_qos_profile_name	1720
6.48.2.5 domain_entity_qos_library_name	1721
6.48.2.6 domain_entity_qos_profile_name	1721
6.49 DDS_DomainParticipantFactoryQos Struct Reference	1721
6.49.1 Detailed Description	1722
6.49.2 Field Documentation	1722
6.49.2.1 entity_factory	1722
6.49.2.2 resource_limits	1722
6.49.2.3 profile	1722
6.49.2.4 logging	1722
6.49.2.5 monitoring	1723
6.50 DDS_DomainParticipantListener Struct Reference	1723
6.50.1 Detailed Description	1723
6.50.2 Field Documentation	1724
6.50.2.1 as_listener	1724
6.50.2.2 as_topiclistener	1724
6.50.2.3 as_publisherlistener	1725
6.50.2.4 as_subscriberlistener	1725
6.50.2.5 on_invalid_local_status_advance_notice	1725
6.51 DDS_DomainParticipantProtocolStatus Struct Reference	1725
6.51.1 Detailed Description	1726
6.51.2 Field Documentation	1726
6.51.2.1 corrupted_rtps_message_count	1726
6.51.2.2 corrupted_rtps_message_count_change	1726
6.51.2.3 last_corrupted_message_timestamp	1726
6.52 DDS_DomainParticipantQos Struct Reference	1726
6.52.1 Detailed Description	1727
6.52.2 Field Documentation	1728
6.52.2.1 user_data	1728
6.52.2.2 entity_factory	1728
6.52.2.3 wire_protocol	1729
6.52.2.4 transport_builtin	1729
6.52.2.5 default_unicast	1729

6.52.2.6 discovery	1729
6.52.2.7 resource_limits	1729
6.52.2.8 event	1729
6.52.2.9 receiver_pool	1730
6.52.2.10 database	1730
6.52.2.11 discovery_config	1730
6.52.2.12 property	1730
6.52.2.13 participant_name	1730
6.52.2.14 multicast_mapping	1730
6.52.2.15 service	1731
6.52.2.16 partition	1731
6.52.2.17 type_support	1731
6.53 DDS_DomainParticipantResourceLimitsQosPolicy Struct Reference	1731
6.53.1 Detailed Description	1734
6.53.2 Field Documentation	1735
6.53.2.1 local_writer_allocation	1735
6.53.2.2 local_reader_allocation	1736
6.53.2.3 local_publisher_allocation	1736
6.53.2.4 local_subscriber_allocation	1736
6.53.2.5 local_topic_allocation	1737
6.53.2.6 remote_writer_allocation	1737
6.53.2.7 remote_reader_allocation	1737
6.53.2.8 remote_participant_allocation	1738
6.53.2.9 matching_writer_reader_pair_allocation	1738
6.53.2.10 matching_reader_writer_pair_allocation	1738
6.53.2.11 ignored_entity_allocation	1739
6.53.2.12 content_filtered_topic_allocation	1739
6.53.2.13 content_filter_allocation	1739
6.53.2.14 read_condition_allocation	1740
6.53.2.15 query_condition_allocation	1740
6.53.2.16 outstanding_asynchronous_sample_allocation	1740
6.53.2.17 flow_controller_allocation	1741
6.53.2.18 local_writer_hash_buckets	1741
6.53.2.19 local_reader_hash_buckets	1741
6.53.2.20 local_publisher_hash_buckets	1742
6.53.2.21 local_subscriber_hash_buckets	1742
6.53.2.22 local_topic_hash_buckets	1742
6.53.2.23 remote_writer_hash_buckets	1743
6.53.2.24 remote_reader_hash_buckets	1743

6.53.2.25 remote_participant_hash_buckets	1743
6.53.2.26 matching_writer_reader_pair_hash_buckets	1744
6.53.2.27 matching_reader_writer_pair_hash_buckets	1744
6.53.2.28 ignored_entity_hash_buckets	1744
6.53.2.29 content_filtered_topic_hash_buckets	1745
6.53.2.30 content_filter_hash_buckets	1745
6.53.2.31 flow_controller_hash_buckets	1745
6.53.2.32 max_gather_destinations	1746
6.53.2.33 participant_user_data_max_length	1746
6.53.2.34 topic_data_max_length	1746
6.53.2.35 publisher_group_data_max_length	1747
6.53.2.36 subscriber_group_data_max_length	1747
6.53.2.37 writer_user_data_max_length	1747
6.53.2.38 reader_user_data_max_length	1748
6.53.2.39 max_partitions	1748
6.53.2.40 max_partition_cumulative_characters	1749
6.53.2.41 type_code_max_serialized_length	1749
6.53.2.42 type_object_max_serialized_length	1750
6.53.2.43 serialized_type_object_dynamic_allocation_threshold	1750
6.53.2.44 type_object_max_deserialized_length	1751
6.53.2.45 deserialized_type_object_dynamic_allocation_threshold	1751
6.53.2.46 contentfilter_property_max_length	1752
6.53.2.47 channel_seq_max_length	1752
6.53.2.48 channel_filter_expression_max_length	1752
6.53.2.49 participant_property_list_max_length	1753
6.53.2.50 participant_property_string_max_length	1753
6.53.2.51 writer_property_list_max_length	1753
6.53.2.52 writer_property_string_max_length	1754
6.53.2.53 reader_property_list_max_length	1754
6.53.2.54 reader_property_string_max_length	1754
6.53.2.55 max_endpoint_groups	1755
6.53.2.56 max_endpoint_group_cumulative_characters	1755
6.53.2.57 transport_info_list_max_length	1755
6.53.2.58 ignored_entity_replacement_kind	1756
6.53.2.59 remote_topic_query_allocation	1756
6.53.2.60 remote_topic_query_hash_buckets	1757
6.53.2.61 writer_data_tag_list_max_length	1757
6.53.2.62 writer_data_tag_string_max_length	1757
6.53.2.63 reader_data_tag_list_max_length	1758

6.53.2.64 reader_data_tag_string_max_length	1758
6.53.2.65 shmem_ref_transfer_mode_max_segments	1758
6.54 DDS_DomainParticipantSeq Struct Reference	1759
6.54.1 Detailed Description	1759
6.55 DDS_DoubleSeq Struct Reference	1759
6.55.1 Detailed Description	1759
6.56 DDS_DurabilityQosPolicy Struct Reference	1759
6.56.1 Detailed Description	1760
6.56.2 Usage	1760
6.56.2.1 Transient and Persistent Durability	1761
6.56.3 Compatibility	1762
6.56.4 Field Documentation	1762
6.56.4.1 kind	1762
6.56.4.2 direct_communication	1762
6.56.4.3 writer_depth	1763
6.56.4.4 storage_settings	1763
6.57 DDS_DurabilityServiceQosPolicy Struct Reference	1764
6.57.1 Detailed Description	1764
6.57.2 Usage	1765
6.57.3 Field Documentation	1765
6.57.3.1 service_cleanup_delay	1765
6.57.3.2 history_kind	1765
6.57.3.3 history_depth	1766
6.57.3.4 max_samples	1766
6.57.3.5 max_instances	1766
6.57.3.6 max_samples_per_instance	1767
6.58 DDS_Duration_t Struct Reference	1767
6.58.1 Detailed Description	1767
6.58.2 Field Documentation	1767
6.58.2.1 sec	1768
6.58.2.2 nanosec	1768
6.59 DDS_DynamicData Struct Reference	1768
6.59.1 Detailed Description	1768
6.59.2 Member Names and IDs	1769
6.59.2.1 Getting and Setting Union Discriminator Values	1769
6.59.2.2 Hierarchical Member Names	1770
6.59.3 Arrays and Sequences	1770
6.59.4 Available Functionality	1770
6.59.4.1 Lifecycle and Utility Methods	1771

6.59.4.2 Getters and Setters	1771
6.59.4.3 Query and Iteration	1774
6.59.4.4 Type/Object Association	1774
6.59.4.5 Performance Considerations	1775
6.60 DDS_DynamicDataInfo Struct Reference	1775
6.60.1 Detailed Description	1776
6.60.2 Field Documentation	1776
6.60.2.1 member_count	1776
6.60.2.2 stored_size	1776
6.61 DDS_DynamicDataJsonParserProperties_t Struct Reference	1776
6.61.1 Detailed Description	1777
6.62 DDS_DynamicDataMemberInfo Struct Reference	1777
6.62.1 Detailed Description	1777
6.62.2 Field Documentation	1777
6.62.2.1 member_id	1778
6.62.2.2 member_name	1778
6.62.2.3 member_exists	1778
6.62.2.4 member_kind	1779
6.62.2.5 element_count	1779
6.62.2.6 element_kind	1779
6.63 DDS_DynamicDataProperty_t Struct Reference	1779
6.63.1 Detailed Description	1779
6.63.2 Field Documentation	1780
6.63.2.1 buffer_initial_size	1780
6.63.2.2 buffer_max_size	1780
6.64 DDS_DynamicDataSeq Struct Reference	1781
6.64.1 Detailed Description	1781
6.65 DDS_DynamicDataTypeProperty_t Struct Reference	1781
6.65.1 Detailed Description	1781
6.65.2 Field Documentation	1781
6.65.2.1 data	1782
6.65.2.2 serialization	1782
6.66 DDS_DynamicDataTypeSerializationProperty_t Struct Reference	1782
6.66.1 Detailed Description	1782
6.66.2 Field Documentation	1782
6.66.2.1 use_42e_compatible_alignment	1783
6.66.2.2 max_size_serialized	1783
6.66.2.3 min_size_serialized	1783
6.66.2.4 trim_to_size	1783

6.66.2.5 skip_deserialization	1784
6.67 DDS_EndpointGroup_t Struct Reference	1784
6.67.1 Detailed Description	1784
6.67.2 Field Documentation	1784
6.67.2.1 role_name	1785
6.67.2.2 quorum_count	1785
6.68 DDS_EndpointGroupSeq Struct Reference	1785
6.68.1 Detailed Description	1785
6.69 DDS_EndpointTrustAlgorithmInfo Struct Reference	1786
6.69.1 Detailed Description	1786
6.69.2 Field Documentation	1786
6.69.2.1 interceptor	1786
6.70 DDS_EndpointTrustInterceptorAlgorithmInfo Struct Reference	1786
6.70.1 Detailed Description	1786
6.70.2 Field Documentation	1787
6.70.2.1 required_mask	1787
6.70.2.2 supported_mask	1787
6.71 DDS_EndpointTrustProtectionInfo Struct Reference	1787
6.71.1 Detailed Description	1787
6.71.2 Field Documentation	1787
6.71.2.1 bitmask	1788
6.71.2.2 plugin_bitmask	1788
6.72 DDS_EntityFactoryQosPolicy Struct Reference	1788
6.72.1 Detailed Description	1788
6.72.2 Usage	1789
6.72.3 Field Documentation	1789
6.72.3.1 autoenable_created_entities	1789
6.73 DDS_EntityNameQosPolicy Struct Reference	1790
6.73.1 Detailed Description	1790
6.73.2 Usage	1790
6.73.3 Field Documentation	1790
6.73.3.1 name	1791
6.73.3.2 role_name	1791
6.74 DDS_EnumMember Struct Reference	1791
6.74.1 Detailed Description	1792
6.74.2 Field Documentation	1792
6.74.2.1 name	1792
6.74.2.2 ordinal	1792
6.75 DDS_EnumMemberSeq Struct Reference	1792

6.75.1 Detailed Description	1792
6.76 DDS_EventQosPolicy Struct Reference	1793
6.76.1 Detailed Description	1793
6.76.2 Field Documentation	1793
6.76.2.1 thread	1794
6.76.2.2 initial_count	1794
6.76.2.3 max_count	1795
6.77 DDS_ExpressionProperty Struct Reference	1795
6.77.1 Detailed Description	1795
6.77.2 Field Documentation	1795
6.77.2.1 key_only_filter	1795
6.77.2.2 writer_side_filter_optimization	1796
6.78 DDS_FilterSampleInfo Struct Reference	1796
6.78.1 Detailed Description	1796
6.78.2 Field Documentation	1796
6.78.2.1 related_sample_identity	1797
6.78.2.2 related_source_guid	1797
6.78.2.3 related_reader_guid	1797
6.78.2.4 sample_identity	1798
6.78.2.5 source_guid	1798
6.79 DDS_FloatSeq Struct Reference	1798
6.79.1 Detailed Description	1798
6.80 DDS_FlowControllerProperty_t Struct Reference	1799
6.80.1 Detailed Description	1799
6.80.2 Field Documentation	1799
6.80.2.1 scheduling_policy	1800
6.80.2.2 token_bucket	1800
6.81 DDS_FlowControllerTokenBucketProperty_t Struct Reference	1800
6.81.1 Detailed Description	1801
6.81.2 Field Documentation	1801
6.81.2.1 max_tokens	1801
6.81.2.2 tokens_added_per_period	1802
6.81.2.3 tokens_leaked_per_period	1802
6.81.2.4 period	1802
6.81.2.5 bytes_per_token	1803
6.82 DDS_GroupDataQosPolicy Struct Reference	1803
6.82.1 Detailed Description	1803
6.82.2 Usage	1804
6.82.3 Field Documentation	1804

6.82.3.1 value	1804
6.83 DDS_GUID_t Struct Reference	1804
6.83.1 Detailed Description	1805
6.83.2 Field Documentation	1805
6.83.2.1 value	1805
6.84 DDS_HistoryQosPolicy Struct Reference	1805
6.84.1 Detailed Description	1806
6.84.2 Usage	1806
6.84.3 Consistency	1807
6.84.4 Field Documentation	1807
6.84.4.1 kind	1807
6.84.4.2 depth	1808
6.85 DDS_InconsistentTopicStatus Struct Reference	1808
6.85.1 Detailed Description	1809
6.85.2 Field Documentation	1809
6.85.2.1 total_count	1809
6.85.2.2 total_count_change	1809
6.86 DDS_InstanceHandleSeq Struct Reference	1809
6.86.1 Detailed Description	1810
6.87 DDS_Int8Seq Struct Reference	1810
6.87.1 Detailed Description	1810
6.88 DDS_InvalidLocalAdvanceNoticeStatus Struct Reference	1810
6.88.1 Detailed Description	1811
6.88.2 Field Documentation	1811
6.88.2.1 expiration_time	1811
6.88.2.2 expiration_code	1811
6.88.2.3 expiration_message	1811
6.89 DDS_KeyedOctets Struct Reference	1811
6.89.1 Detailed Description	1812
6.90 DDS_KeyedOctetsSeq Struct Reference	1812
6.90.1 Detailed Description	1812
6.91 DDS_KeyedOctetsTypeSupport Struct Reference	1812
6.91.1 Detailed Description	1812
6.92 DDS_KeyedString Struct Reference	1812
6.92.1 Detailed Description	1813
6.93 DDS_KeyedStringSeq Struct Reference	1813
6.93.1 Detailed Description	1813
6.94 DDS_KeyedStringTypeSupport Struct Reference	1813
6.94.1 Detailed Description	1813

6.95 DDS_LatencyBudgetQosPolicy Struct Reference	1813
6.95.1 Detailed Description	1814
6.95.2 Usage	1814
6.95.3 Compatibility	1814
6.95.4 Field Documentation	1815
6.95.4.1 duration	1815
6.96 DDS_LifespanQosPolicy Struct Reference	1815
6.96.1 Detailed Description	1815
6.96.2 Usage	1816
6.96.3 Field Documentation	1816
6.96.3.1 duration	1816
6.97 DDS_Listener Struct Reference	1816
6.97.1 Detailed Description	1817
6.97.2 Access to Plain Communication Status	1817
6.97.3 Access to Read Communication Status	1818
6.97.4 Operations Allowed in Listener Callbacks	1818
6.97.5 Best Practices with Listeners	1818
6.97.6 Field Documentation	1819
6.97.6.1 listener_data	1819
6.98 DDS_LivelinessChangedStatus Struct Reference	1820
6.98.1 Detailed Description	1820
6.98.2 Field Documentation	1821
6.98.2.1 alive_count	1821
6.98.2.2 not_alive_count	1821
6.98.2.3 alive_count_change	1821
6.98.2.4 not_alive_count_change	1821
6.98.2.5 last_publication_handle	1822
6.99 DDS_LivelinessLostStatus Struct Reference	1822
6.99.1 Detailed Description	1822
6.99.2 Field Documentation	1822
6.99.2.1 total_count	1823
6.99.2.2 total_count_change	1823
6.100 DDS_LivelinessQosPolicy Struct Reference	1823
6.100.1 Detailed Description	1823
6.100.2 Usage	1824
6.100.3 Compatibility	1825
6.100.4 Field Documentation	1825
6.100.4.1 kind	1825
6.100.4.2 lease_duration	1826

6.100.4.3 assertions_per_lease_duration	1826
6.101 DDS_Locator_t Struct Reference	1826
6.101.1 Detailed Description	1827
6.101.2 Field Documentation	1827
6.101.2.1 kind	1827
6.101.2.2 port	1827
6.101.2.3 address	1827
6.102 DDS_LocatorFilter_t Struct Reference	1828
6.102.1 Detailed Description	1828
6.102.2 Field Documentation	1828
6.102.2.1 locators	1828
6.102.2.2 filter_expression	1828
6.103 DDS_LocatorFilterQosPolicy Struct Reference	1829
6.103.1 Detailed Description	1829
6.103.2 Field Documentation	1829
6.103.2.1 locator_filters	1829
6.103.2.2 filter_name	1830
6.104 DDS_LocatorFilterSeq Struct Reference	1830
6.104.1 Detailed Description	1830
6.105 DDS_LocatorSeq Struct Reference	1830
6.105.1 Detailed Description	1831
6.106 DDS_LoggingQosPolicy Struct Reference	1831
6.106.1 Detailed Description	1831
6.106.2 Field Documentation	1832
6.106.2.1 verbosity	1832
6.106.2.2 category	1832
6.106.2.3 print_format	1832
6.106.2.4 output_file	1833
6.106.2.5 output_file_suffix	1833
6.106.2.6 max_bytes_per_file	1834
6.106.2.7 max_files	1834
6.107 DDS_LongDoubleSeq Struct Reference	1835
6.107.1 Detailed Description	1835
6.108 DDS_LongLongSeq Struct Reference	1835
6.108.1 Detailed Description	1835
6.109 DDS_LongSeq Struct Reference	1835
6.109.1 Detailed Description	1836
6.110 DDS_MonitoringDedicatedParticipantSettings Struct Reference	1836
6.110.1 Detailed Description	1836

6.110.2 Field Documentation	1836
6.110.2.1 enable	1837
6.110.2.2 domain_id	1837
6.110.2.3 participant_qos_profile_name	1837
6.110.2.4 collector_initial_peers	1838
6.111 DDS_MonitoringDistributionSettings Struct Reference	1838
6.111.1 Detailed Description	1839
6.111.2 Field Documentation	1839
6.111.2.1 dedicated_participant	1839
6.111.2.2 publisher_qos_profile_name	1839
6.111.2.3 event_settings	1840
6.111.2.4 periodic_settings	1840
6.111.2.5 logging_settings	1840
6.112 DDS_MonitoringEventDistributionSettings Struct Reference	1840
6.112.1 Detailed Description	1841
6.112.2 Field Documentation	1841
6.112.2.1 concurrency_level	1842
6.112.2.2 datawriter_qos_profile_name	1842
6.112.2.3 thread	1843
6.112.2.4 publication_period	1843
6.113 DDS_MonitoringLoggingDistributionSettings Struct Reference	1843
6.113.1 Detailed Description	1844
6.113.2 Field Documentation	1844
6.113.2.1 concurrency_level	1845
6.113.2.2 max_historical_logs	1845
6.113.2.3 datawriter_qos_profile_name	1846
6.113.2.4 thread	1846
6.113.2.5 publication_period	1846
6.114 DDS_MonitoringLoggingForwardingSettings Struct Reference	1847
6.114.1 Detailed Description	1847
6.114.2 Field Documentation	1847
6.114.2.1 middleware_forwarding_level	1847
6.114.2.2 security_event_forwarding_level	1848
6.114.2.3 service_forwarding_level	1848
6.114.2.4 user_forwarding_level	1848
6.115 DDS_MonitoringMetricSelection Struct Reference	1849
6.115.1 Detailed Description	1849
6.115.2 Field Documentation	1849
6.115.2.1 resource_selection	1849

6.115.2.2 enabled_metrics_selection	1850
6.115.2.3 disabled_metrics_selection	1851
6.116 DDS_MonitoringMetricSelectionSeq Struct Reference	1851
6.116.1 Detailed Description	1851
6.117 DDS_MonitoringPeriodicDistributionSettings Struct Reference	1852
6.117.1 Detailed Description	1852
6.117.2 Field Documentation	1852
6.117.2.1 datawriter_qos_profile_name	1852
6.117.2.2 thread	1853
6.117.2.3 polling_period	1853
6.118 DDS_MonitoringQosPolicy Struct Reference	1853
6.118.1 Detailed Description	1854
6.118.2 Field Documentation	1854
6.118.2.1 enable	1854
6.118.2.2 application_name	1854
6.118.2.3 distribution_settings	1855
6.118.2.4 telemetry_data	1855
6.119 DDS_MonitoringTelemetryData Struct Reference	1855
6.119.1 Detailed Description	1855
6.119.2 Field Documentation	1855
6.119.2.1 metrics	1856
6.119.2.2 logs	1856
6.120 DDS_MultiChannelQosPolicy Struct Reference	1856
6.120.1 Detailed Description	1857
6.120.2 Usage	1857
6.120.3 Field Documentation	1858
6.120.3.1 channels	1858
6.120.3.2 filter_name	1858
6.121 DDS_Octets Struct Reference	1859
6.121.1 Detailed Description	1860
6.122 DDS_OctetSeq Struct Reference	1860
6.122.1 Detailed Description	1860
6.123 DDS_OctetsSeq Struct Reference	1860
6.123.1 Detailed Description	1860
6.124 DDS_OctetsTypeSupport Struct Reference	1860
6.124.1 Detailed Description	1861
6.125 DDS_OfferedDeadlineMissedStatus Struct Reference	1861
6.125.1 Detailed Description	1861
6.125.2 Field Documentation	1861

6.125.2.1 total_count	1861
6.125.2.2 total_count_change	1862
6.125.2.3 last_instance_handle	1862
6.126 DDS_OfferedIncompatibleQosStatus Struct Reference	1862
6.126.1 Detailed Description	1862
6.126.2 Field Documentation	1863
6.126.2.1 total_count	1863
6.126.2.2 total_count_change	1863
6.126.2.3 last_policy_id	1863
6.126.2.4 policies	1863
6.127 DDS_OwnershipQosPolicy Struct Reference	1863
6.127.1 Detailed Description	1864
6.127.2 Usage	1864
6.127.2.1 SHARED ownership	1864
6.127.2.2 EXCLUSIVE ownership	1865
6.127.3 Compatibility	1865
6.127.4 Relationship between registration, liveliness and ownership	1866
6.127.4.1 Ownership Resolution on Redundant Systems	1866
6.127.4.2 Detection of Loss in Topological Connectivity	1867
6.127.4.3 Semantic Difference between unregister_instance and dispose	1868
6.127.5 Field Documentation	1868
6.127.5.1 kind	1868
6.128 DDS_OwnershipStrengthQosPolicy Struct Reference	1868
6.128.1 Detailed Description	1869
6.128.2 Field Documentation	1869
6.128.2.1 value	1869
6.129 DDS_ParticipantBuiltinTopicData Struct Reference	1870
6.129.1 Detailed Description	1871
6.129.2 Field Documentation	1871
6.129.2.1 key	1871
6.129.2.2 user_data	1871
6.129.2.3 property	1871
6.129.2.4 rtps_protocol_version	1871
6.129.2.5 rtps_vendor_id	1872
6.129.2.6 dds_builtin_endpoints	1872
6.129.2.7 available_builtin_endpoints_ext	1873
6.129.2.8 default_unicast_locators	1873
6.129.2.9 product_version	1873
6.129.2.10 participant_name	1873

6.129.2.11 domain_id	1873
6.129.2.12 transport_info	1874
6.129.2.13 reachability_lease_duration	1874
6.129.2.14 partition	1874
6.129.2.15 trust_protection_info	1874
6.129.2.16 trust_algorithm_info	1875
6.129.2.17 partial_configuration	1875
6.130 DDS_ParticipantBuiltinTopicDataSeq Struct Reference	1876
6.130.1 Detailed Description	1876
6.131 DDS_ParticipantBuiltinTopicDataTypeSupport Struct Reference	1876
6.131.1 Detailed Description	1876
6.132 DDS_ParticipantTrustAlgorithmInfo Struct Reference	1876
6.132.1 Detailed Description	1877
6.132.2 Field Documentation	1877
6.132.2.1 signature	1877
6.132.2.2 key_establishment	1877
6.132.2.3 interceptor	1877
6.133 DDS_ParticipantTrustInterceptorAlgorithmInfo Struct Reference	1877
6.133.1 Detailed Description	1878
6.133.2 Field Documentation	1878
6.133.2.1 supported_mask	1878
6.133.2.2 builtin_endpoints_required_mask	1878
6.133.2.3 builtin_kx_endpoints_required_mask	1878
6.133.2.4 user_endpoints_default_required_mask	1878
6.134 DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo Struct Reference	1879
6.134.1 Detailed Description	1879
6.134.2 Field Documentation	1879
6.134.2.1 shared_secret	1879
6.135 DDS_ParticipantTrustProtectionInfo Struct Reference	1879
6.135.1 Detailed Description	1879
6.135.2 Field Documentation	1880
6.135.2.1 bitmask	1880
6.135.2.2 plugin_bitmask	1880
6.136 DDS_ParticipantTrustSignatureAlgorithmInfo Struct Reference	1880
6.136.1 Detailed Description	1880
6.136.2 Field Documentation	1881
6.136.2.1 trust_chain	1881
6.136.2.2 message_auth	1881
6.137 DDS_PartitionQosPolicy Struct Reference	1881

6.137.1 Detailed Description	1881
6.137.2 Usage	1882
6.137.3 Field Documentation	1883
6.137.3.1 name	1883
6.138 DDS_PersistentStorageSettings Struct Reference	1883
6.138.1 Detailed Description	1884
6.138.2 Field Documentation	1885
6.138.2.1 enable	1885
6.138.2.2 file_name	1885
6.138.2.3 trace_file_name	1886
6.138.2.4 journal_kind	1886
6.138.2.5 synchronization_kind	1886
6.138.2.6 vacuum	1887
6.138.2.7 restore	1887
6.138.2.8 writer_instance_cache_allocation	1888
6.138.2.9 writer_sample_cache_allocation	1888
6.138.2.10 writer_memory_state	1889
6.138.2.11 reader_checkpoint_frequency	1889
6.139 DDS_PresentationQosPolicy Struct Reference	1890
6.139.1 Detailed Description	1890
6.139.2 Usage	1891
6.139.3 Compatibility	1892
6.139.4 Field Documentation	1893
6.139.4.1 access_scope	1893
6.139.4.2 coherent_access	1893
6.139.4.3 ordered_access	1894
6.139.4.4 drop_incomplete_coherent_set	1894
6.140 DDS_PrintFormatProperty Struct Reference	1894
6.140.1 Detailed Description	1895
6.140.2 Field Documentation	1895
6.140.2.1 kind	1895
6.140.2.2 pretty_print	1896
6.140.2.3 enum_as_int	1896
6.140.2.4 include_root_elements	1896
6.141 DDS_ProductVersion_t Struct Reference	1897
6.141.1 Detailed Description	1897
6.141.2 Field Documentation	1897
6.141.2.1 major	1897
6.141.2.2 minor	1897

6.141.2.3 release	1898
6.141.2.4 revision	1898
6.142 DDS_ProfileQosPolicy Struct Reference	1898
6.142.1 Detailed Description	1898
6.142.2 Field Documentation	1899
6.142.2.1 string_profile	1899
6.142.2.2 url_profile	1899
6.142.2.3 ignore_user_profile	1899
6.142.2.4 ignore_environment_profile	1900
6.142.2.5 ignore_resource_profile	1900
6.143 DDS_Property_t Struct Reference	1900
6.143.1 Detailed Description	1901
6.143.2 Field Documentation	1901
6.143.2.1 name	1901
6.143.2.2 value	1901
6.143.2.3 propagate	1901
6.144 DDS_PropertyQosPolicy Struct Reference	1901
6.144.1 Detailed Description	1902
6.144.2 Usage	1902
6.144.2.1 Reasons for Using the PropertyQosPolicy	1903
6.144.3 Field Documentation	1903
6.144.3.1 value	1903
6.145 DDS_PropertySeq Struct Reference	1903
6.145.1 Detailed Description	1904
6.146 DDS_ProtocolVersion_t Struct Reference	1904
6.146.1 Detailed Description	1904
6.146.2 Field Documentation	1904
6.146.2.1 major	1904
6.146.2.2 minor	1904
6.147 DDS_PublicationBuiltinTopicData Struct Reference	1905
6.147.1 Detailed Description	1906
6.147.2 Field Documentation	1907
6.147.2.1 key	1907
6.147.2.2 participant_key	1907
6.147.2.3 topic_name	1907
6.147.2.4 type_name	1907
6.147.2.5 durability	1908
6.147.2.6 durability_service	1908
6.147.2.7 deadline	1908

6.147.2.8 latency_budget	1908
6.147.2.9 liveliness	1908
6.147.2.10 reliability	1908
6.147.2.11 lifespan	1909
6.147.2.12 user_data	1909
6.147.2.13 ownership	1909
6.147.2.14 ownership_strength	1909
6.147.2.15 destination_order	1909
6.147.2.16 presentation	1910
6.147.2.17 partition	1910
6.147.2.18 topic_data	1910
6.147.2.19 group_data	1910
6.147.2.20 representation	1910
6.147.2.21 data_tags	1910
6.147.2.22 type_code	1911
6.147.2.23 publisher_key	1911
6.147.2.24 property	1911
6.147.2.25 unicast_locators	1911
6.147.2.26 virtual_guid	1911
6.147.2.27 service	1912
6.147.2.28 rtps_protocol_version	1912
6.147.2.29 rtps_vendor_id	1912
6.147.2.30 product_version	1912
6.147.2.31 locator_filter	1912
6.147.2.32 disable_positive_acks	1912
6.147.2.33 publication_name	1913
6.147.2.34 trust_protection_info	1913
6.147.2.35 trust_algorithm_info	1913
6.147.2.36 related_datareader_key	1913
6.148 DDS_PublicationBuiltinTopicDataSeq Struct Reference	1914
6.148.1 Detailed Description	1914
6.149 DDS_PublicationBuiltinTopicDataTypeSupport Struct Reference	1914
6.149.1 Detailed Description	1914
6.150 DDS_PublicationMatchedStatus Struct Reference	1914
6.150.1 Detailed Description	1915
6.150.2 Field Documentation	1915
6.150.2.1 total_count	1916
6.150.2.2 total_count_change	1916
6.150.2.3 current_count	1916

6.150.2.4 current_count_peak	1916
6.150.2.5 current_count_change	1916
6.150.2.6 last_subscription_handle	1917
6.151 DDS_PublisherListener Struct Reference	1917
6.151.1 Detailed Description	1917
6.151.2 Field Documentation	1918
6.151.2.1 as_datawriterlistener	1918
6.152 DDS_PublisherQos Struct Reference	1918
6.152.1 Detailed Description	1918
6.152.2 Field Documentation	1919
6.152.2.1 presentation	1919
6.152.2.2 partition	1919
6.152.2.3 group_data	1919
6.152.2.4 entity_factory	1919
6.152.2.5 asynchronous_publisher	1919
6.152.2.6 publisher_name	1920
6.153 DDS_PublisherSeq Struct Reference	1920
6.153.1 Detailed Description	1920
6.154 DDS_PublishModeQosPolicy Struct Reference	1920
6.154.1 Detailed Description	1921
6.154.2 Usage	1921
6.154.3 Field Documentation	1922
6.154.3.1 kind	1922
6.154.3.2 flow_controller_name	1922
6.154.3.3 priority	1924
6.155 DDS_QosPolicyCount Struct Reference	1924
6.155.1 Detailed Description	1925
6.155.2 Field Documentation	1925
6.155.2.1 policy_id	1925
6.155.2.2 count	1925
6.156 DDS_QosPolicyCountSeq Struct Reference	1925
6.156.1 Detailed Description	1925
6.157 DDS_QosPrintFormat Struct Reference	1926
6.157.1 Detailed Description	1926
6.157.2 Field Documentation	1926
6.157.2.1 is_standalone	1926
6.157.2.2 print_private	1927
6.157.2.3 indent	1927
6.158 DDS_QueryConditionParams Struct Reference	1928

6.158.1 Detailed Description	1928
6.158.2 Field Documentation	1928
6.158.2.1 as_readconditionparams	1929
6.158.2.2 query_expression	1929
6.158.2.3 query_parameters	1929
6.159 DDS_ReadConditionParams Struct Reference	1929
6.159.1 Detailed Description	1929
6.159.2 Field Documentation	1930
6.159.2.1 sample_states	1930
6.159.2.2 view_states	1930
6.159.2.3 instance_states	1930
6.159.2.4 stream_kinds	1930
6.160 DDS_ReaderDataLifecycleQosPolicy Struct Reference	1930
6.160.1 Detailed Description	1931
6.160.2 Field Documentation	1932
6.160.2.1 autopurge_nowriter_samples_delay	1932
6.160.2.2 autopurge_disposed_samples_delay	1932
6.160.2.3 autopurge_disposed_instances_delay	1933
6.160.2.4 autopurge_nowriter_instances_delay	1933
6.161 DDS_ReceiverPoolQosPolicy Struct Reference	1934
6.161.1 Detailed Description	1934
6.161.2 Usage	1934
6.161.3 Field Documentation	1935
6.161.3.1 thread	1935
6.161.3.2 buffer_size	1935
6.161.3.3 buffer_alignment	1936
6.162 DDS_ReliabilityQosPolicy Struct Reference	1936
6.162.1 Detailed Description	1937
6.162.2 Usage	1937
6.162.3 Compatibility	1938
6.162.4 Field Documentation	1939
6.162.4.1 kind	1939
6.162.4.2 max_blocking_time	1939
6.162.4.3 acknowledgment_kind	1940
6.162.4.4 instance_state_consistency_kind	1940
6.163 DDS_ReliableReaderActivityChangedStatus Struct Reference	1940
6.163.1 Detailed Description	1941
6.163.2 Field Documentation	1941
6.163.2.1 active_count	1941

6.163.2.2 inactive_count	1942
6.163.2.3 active_count_change	1942
6.163.2.4 inactive_count_change	1942
6.163.2.5 last_instance_handle	1942
6.164 DDS_ReliableWriterCacheChangedStatus Struct Reference	1942
6.164.1 Detailed Description	1943
6.164.2 Field Documentation	1943
6.164.2.1 empty_reliable_writer_cache	1944
6.164.2.2 full_reliable_writer_cache	1944
6.164.2.3 low_watermark_reliable_writer_cache	1944
6.164.2.4 high_watermark_reliable_writer_cache	1944
6.164.2.5 unacknowledged_sample_count	1945
6.164.2.6 unacknowledged_sample_count_peak	1945
6.164.2.7 replaced_unacknowledged_sample_count	1945
6.165 DDS_ReliableWriterCacheEventCount Struct Reference	1945
6.165.1 Detailed Description	1945
6.165.2 Field Documentation	1946
6.165.2.1 total_count	1946
6.165.2.2 total_count_change	1946
6.166 DDS_RequestedDeadlineMissedStatus Struct Reference	1946
6.166.1 Detailed Description	1946
6.166.2 Field Documentation	1946
6.166.2.1 total_count	1947
6.166.2.2 total_count_change	1947
6.166.2.3 last_instance_handle	1947
6.167 DDS_RequestedIncompatibleQosStatus Struct Reference	1947
6.167.1 Detailed Description	1948
6.167.2 Field Documentation	1948
6.167.2.1 total_count	1948
6.167.2.2 total_count_change	1948
6.167.2.3 last_policy_id	1949
6.167.2.4 policies	1949
6.168 DDS_ResourceLimitsQosPolicy Struct Reference	1949
6.168.1 Detailed Description	1950
6.168.2 Usage	1950
6.168.3 Consistency	1951
6.168.4 Field Documentation	1951
6.168.4.1 max_samples	1951
6.168.4.2 max_instances	1952

6.168.4.3 max_samples_per_instance	1952
6.168.4.4 initial_samples	1952
6.168.4.5 initial_instances	1953
6.168.4.6 instance_hash_buckets	1953
6.169 DDS_RTPS_EntityId_t Struct Reference	1953
6.169.1 Detailed Description	1953
6.170 DDS_RTPS_GUID_t Struct Reference	1954
6.170.1 Detailed Description	1954
6.171 DDS_RtpsReliableReaderProtocol_t Struct Reference	1954
6.171.1 Detailed Description	1955
6.171.2 Field Documentation	1955
6.171.2.1 min_heartbeat_response_delay	1955
6.171.2.2 max_heartbeat_response_delay	1956
6.171.2.3 heartbeat_suppression_duration	1956
6.171.2.4 nack_period	1957
6.171.2.5 receive_window_size	1957
6.171.2.6 round_trip_time	1958
6.171.2.7 app_ack_period	1958
6.171.2.8 min_app_ack_response_keep_duration	1959
6.171.2.9 samples_per_app_ack	1959
6.172 DDS_RtpsReliableWriterProtocol_t Struct Reference	1960
6.172.1 Detailed Description	1961
6.172.2 Field Documentation	1961
6.172.2.1 low_watermark	1962
6.172.2.2 high_watermark	1962
6.172.2.3 heartbeat_period	1963
6.172.2.4 fast_heartbeat_period	1964
6.172.2.5 late_joiner_heartbeat_period	1964
6.172.2.6 virtual_heartbeat_period	1965
6.172.2.7 samples_per_virtual_heartbeat	1966
6.172.2.8 max_heartbeat_retries	1966
6.172.2.9 inactivate_nonprogressing_readers	1967
6.172.2.10 heartbeats_per_max_samples	1967
6.172.2.11 min_nack_response_delay	1968
6.172.2.12 max_nack_response_delay	1969
6.172.2.13 nack_suppression_duration	1969
6.172.2.14 max_bytes_per_nack_response	1970
6.172.2.15 disable_positive_acks_min_sample_keep_duration	1971
6.172.2.16 disable_positive_acks_max_sample_keep_duration	1971

6.172.2.17	disable_positive_acks_enable_adaptive_sample_keep_duration	1972
6.172.2.18	disable_positive_acks_decrease_sample_keep_duration_factor	1972
6.172.2.19	disable_positive_acks_increase_sample_keep_duration_factor	1973
6.172.2.20	min_send_window_size	1973
6.172.2.21	max_send_window_size	1974
6.172.2.22	send_window_update_period	1975
6.172.2.23	send_window_increase_factor	1976
6.172.2.24	send_window_decrease_factor	1976
6.172.2.25	enable_multicast_periodic_heartbeat	1977
6.172.2.26	multicast_resend_threshold	1978
6.172.2.27	disable_repair_piggyback_heartbeat	1978
6.173	DDS_RtpsWellKnownPorts_t Struct Reference	1978
6.173.1	Detailed Description	1979
6.173.2	Field Documentation	1980
6.173.2.1	port_base	1981
6.173.2.2	domain_id_gain	1981
6.173.2.3	participant_id_gain	1982
6.173.2.4	builtin_multicast_port_offset	1982
6.173.2.5	builtin_unicast_port_offset	1983
6.173.2.6	user_multicast_port_offset	1983
6.173.2.7	user_unicast_port_offset	1983
6.174	DDS_SampleHandler Struct Reference	1983
6.174.1	Detailed Description	1984
6.174.2	Field Documentation	1984
6.174.2.1	handler_data	1984
6.174.2.2	on_new_sample	1984
6.175	DDS_SampleIdentity_t Struct Reference	1984
6.175.1	Detailed Description	1984
6.176	DDS_SampleInfo Struct Reference	1985
6.176.1	Detailed Description	1986
6.176.2	Interpretation of the SampleInfo	1986
6.176.3	Interpretation of the SampleInfo disposed_generation_count and no_writers_generation_count	1987
6.176.4	Interpretation of the SampleInfo sample_rank, generation_rank and absolute_generation_rank	1987
6.176.5	Interpretation of the SampleInfo counters and ranks	1988
6.176.6	Field Documentation	1988
6.176.6.1	sample_state	1989
6.176.6.2	view_state	1989
6.176.6.3	instance_state	1989
6.176.6.4	source_timestamp	1990

6.176.6.5 instance_handle	1990
6.176.6.6 publication_handle	1990
6.176.6.7 disposed_generation_count	1990
6.176.6.8 no_writers_generation_count	1991
6.176.6.9 sample_rank	1991
6.176.6.10 generation_rank	1991
6.176.6.11 absolute_generation_rank	1992
6.176.6.12 valid_data	1992
6.176.6.13 reception_timestamp	1992
6.176.6.14 publication_sequence_number	1993
6.176.6.15 reception_sequence_number	1993
6.176.6.16 original_publication_virtual_guid	1993
6.176.6.17 original_publication_virtual_sequence_number	1993
6.176.6.18 related_original_publication_virtual_guid	1994
6.176.6.19 related_original_publication_virtual_sequence_number	1994
6.176.6.20 flag	1994
6.176.6.21 source_guid	1994
6.176.6.22 related_source_guid	1995
6.176.6.23 related_subscription_guid	1995
6.176.6.24 topic_query_guid	1995
6.176.6.25 coherent_set_info	1995
6.177 DDS_SampleInfoSeq Struct Reference	1996
6.177.1 Detailed Description	1996
6.178 DDS_SampleLostStatus Struct Reference	1996
6.178.1 Detailed Description	1996
6.178.2 Field Documentation	1996
6.178.2.1 total_count	1997
6.178.2.2 total_count_change	1997
6.178.2.3 last_reason	1997
6.179 DDS_SampleRejectedStatus Struct Reference	1997
6.179.1 Detailed Description	1998
6.179.2 Field Documentation	1998
6.179.2.1 total_count	1998
6.179.2.2 total_count_change	1998
6.179.2.3 last_reason	1998
6.179.2.4 last_instance_handle	1999
6.180 DDS_SequenceNumber_t Struct Reference	1999
6.180.1 Detailed Description	1999
6.180.2 Field Documentation	1999

6.180.2.1 high	1999
6.180.2.2 low	1999
6.181 DDS_ServiceQosPolicy Struct Reference	2000
6.181.1 Detailed Description	2000
6.181.2 Field Documentation	2000
6.181.2.1 kind	2000
6.182 DDS_ServiceRequest Struct Reference	2001
6.182.1 Detailed Description	2001
6.182.2 Field Documentation	2001
6.182.2.1 service_id	2001
6.182.2.2 instance_id	2002
6.182.2.3 request_body	2002
6.183 DDS_ServiceRequestAcceptedStatus Struct Reference	2002
6.183.1 Detailed Description	2003
6.183.2 Field Documentation	2003
6.183.2.1 total_count	2003
6.183.2.2 total_count_change	2003
6.183.2.3 current_count	2003
6.183.2.4 current_count_change	2003
6.183.2.5 last_request_handle	2004
6.183.2.6 service_id	2004
6.184 DDS_ServiceRequestSeq Struct Reference	2004
6.184.1 Detailed Description	2004
6.185 DDS_ServiceRequestTypeSupport Struct Reference	2004
6.185.1 Detailed Description	2005
6.186 DDS_ShortSeq Struct Reference	2005
6.186.1 Detailed Description	2005
6.187 DDS_StringSeq Struct Reference	2005
6.187.1 Detailed Description	2006
6.188 DDS_StringTypeSupport Struct Reference	2006
6.188.1 Detailed Description	2007
6.189 DDS_StructMember Struct Reference	2007
6.189.1 Detailed Description	2007
6.189.2 Field Documentation	2007
6.189.2.1 name	2007
6.189.2.2 type	2008
6.189.2.3 is_pointer	2008
6.189.2.4 bits	2008
6.189.2.5 is_key	2008

6.189.2.6 id	2008
6.189.2.7 is_optional	2008
6.190 DDS_StructMemberSeq Struct Reference	2009
6.190.1 Detailed Description	2009
6.191 DDS_SubscriberListener Struct Reference	2009
6.191.1 Detailed Description	2009
6.191.2 Field Documentation	2010
6.191.2.1 as_datareaderlistener	2010
6.191.2.2 on_data_on_readers	2010
6.192 DDS_SubscriberQos Struct Reference	2010
6.192.1 Detailed Description	2010
6.192.2 Field Documentation	2011
6.192.2.1 presentation	2011
6.192.2.2 partition	2011
6.192.2.3 group_data	2011
6.192.2.4 entity_factory	2011
6.192.2.5 subscriber_name	2011
6.193 DDS_SubscriberSeq Struct Reference	2012
6.193.1 Detailed Description	2012
6.194 DDS_SubscriptionBuiltinTopicData Struct Reference	2012
6.194.1 Detailed Description	2014
6.194.2 Field Documentation	2014
6.194.2.1 key	2014
6.194.2.2 participant_key	2014
6.194.2.3 topic_name	2014
6.194.2.4 type_name	2015
6.194.2.5 durability	2015
6.194.2.6 deadline	2015
6.194.2.7 latency_budget	2015
6.194.2.8 liveliness	2015
6.194.2.9 reliability	2016
6.194.2.10 ownership	2016
6.194.2.11 destination_order	2016
6.194.2.12 user_data	2016
6.194.2.13 time_based_filter	2016
6.194.2.14 presentation	2017
6.194.2.15 partition	2017
6.194.2.16 topic_data	2017
6.194.2.17 group_data	2017

6.194.2.18 type_consistency	2017
6.194.2.19 representation	2017
6.194.2.20 data_tags	2018
6.194.2.21 type_code	2018
6.194.2.22 subscriber_key	2018
6.194.2.23 property	2018
6.194.2.24 unicast_locators	2018
6.194.2.25 multicast_locators	2019
6.194.2.26 content_filter_property	2019
6.194.2.27 virtual_guid	2019
6.194.2.28 service	2019
6.194.2.29 rtps_protocol_version	2019
6.194.2.30 rtps_vendor_id	2020
6.194.2.31 product_version	2020
6.194.2.32 disable_positive_acks	2020
6.194.2.33 subscription_name	2020
6.194.2.34 trust_protection_info	2020
6.194.2.35 trust_algorithm_info	2021
6.194.2.36 related_datawriter_key	2021
6.195 DDS_SubscriptionBuiltinTopicDataSeq Struct Reference	2021
6.195.1 Detailed Description	2021
6.196 DDS_SubscriptionBuiltinTopicDataTypeSupport Struct Reference	2021
6.196.1 Detailed Description	2022
6.197 DDS_SubscriptionMatchedStatus Struct Reference	2022
6.197.1 Detailed Description	2022
6.197.2 Field Documentation	2023
6.197.2.1 total_count	2023
6.197.2.2 total_count_change	2023
6.197.2.3 current_count	2023
6.197.2.4 current_count_peak	2024
6.197.2.5 current_count_change	2024
6.197.2.6 last_publication_handle	2024
6.198 DDS_SystemResourceLimitsQosPolicy Struct Reference	2024
6.198.1 Detailed Description	2025
6.198.2 Usage	2025
6.198.3 Field Documentation	2025
6.198.3.1 max_objects_per_thread	2025
6.198.3.2 initial_objects_per_thread	2026
6.199 DDS_Tag Struct Reference	2026

6.199.1 Detailed Description	2026
6.199.2 Field Documentation	2026
6.199.2.1 name	2027
6.199.2.2 value	2027
6.200 DDS_TagSeq Struct Reference	2027
6.200.1 Detailed Description	2027
6.201 DDS_ThreadFactory Struct Reference	2027
6.201.1 Detailed Description	2028
6.201.2 Field Documentation	2028
6.201.2.1 factory_data	2028
6.201.2.2 create_thread	2028
6.201.2.3 delete_thread	2029
6.202 DDS_ThreadSettings_t Struct Reference	2029
6.202.1 Detailed Description	2029
6.202.2 Field Documentation	2029
6.202.2.1 mask	2030
6.202.2.2 priority	2030
6.202.2.3 stack_size	2030
6.202.2.4 cpu_list	2030
6.202.2.5 cpu_rotation	2031
6.203 DDS_Time_t Struct Reference	2031
6.203.1 Detailed Description	2031
6.203.2 Field Documentation	2031
6.203.2.1 sec	2031
6.203.2.2 nanosec	2032
6.204 DDS_TimeBasedFilterQosPolicy Struct Reference	2032
6.204.1 Detailed Description	2032
6.204.2 Usage	2033
6.204.3 Consistency	2033
6.204.4 Field Documentation	2034
6.204.4.1 minimum_separation	2034
6.205 DDS_TopicBuiltinTopicData Struct Reference	2034
6.205.1 Detailed Description	2035
6.205.2 Field Documentation	2036
6.205.2.1 key	2036
6.205.2.2 name	2036
6.205.2.3 type_name	2036
6.205.2.4 durability	2037
6.205.2.5 durability_service	2037

6.205.2.6 deadline	2037
6.205.2.7 latency_budget	2037
6.205.2.8 liveliness	2037
6.205.2.9 reliability	2037
6.205.2.10 transport_priority	2038
6.205.2.11 lifespan	2038
6.205.2.12 destination_order	2038
6.205.2.13 history	2038
6.205.2.14 resource_limits	2038
6.205.2.15 ownership	2038
6.205.2.16 topic_data	2039
6.205.2.17 representation	2039
6.206 DDS_TopicBuiltinTopicDataSeq Struct Reference	2039
6.206.1 Detailed Description	2039
6.207 DDS_TopicBuiltinTopicDataTypeSupport Struct Reference	2039
6.207.1 Detailed Description	2039
6.208 DDS_TopicDataQosPolicy Struct Reference	2040
6.208.1 Detailed Description	2040
6.208.2 Usage	2040
6.208.3 Field Documentation	2041
6.208.3.1 value	2041
6.209 DDS_TopicListener Struct Reference	2041
6.209.1 Detailed Description	2041
6.209.2 Field Documentation	2042
6.209.2.1 as_listener	2042
6.209.2.2 on_inconsistent_topic	2042
6.210 DDS_TopicQos Struct Reference	2042
6.210.1 Detailed Description	2043
6.210.2 Field Documentation	2043
6.210.2.1 topic_data	2043
6.210.2.2 durability	2044
6.210.2.3 durability_service	2044
6.210.2.4 deadline	2044
6.210.2.5 latency_budget	2044
6.210.2.6 liveliness	2044
6.210.2.7 reliability	2044
6.210.2.8 destination_order	2045
6.210.2.9 history	2045
6.210.2.10 resource_limits	2045

6.210.2.11 transport_priority	2045
6.210.2.12 lifespan	2045
6.210.2.13 ownership	2045
6.210.2.14 representation	2046
6.211 DDS_TopicQueryData Struct Reference	2046
6.211.1 Detailed Description	2046
6.211.2 Field Documentation	2046
6.211.2.1 topic_query_selection	2046
6.211.2.2 topic_name	2047
6.211.2.3 original_related_reader_guid	2047
6.212 DDS_TopicQueryDispatchQosPolicy Struct Reference	2047
6.212.1 Detailed Description	2047
6.212.2 Field Documentation	2048
6.212.2.1 enable	2048
6.212.2.2 publication_period	2049
6.212.2.3 samples_per_period	2049
6.213 DDS_TopicQuerySelection Struct Reference	2049
6.213.1 Detailed Description	2050
6.213.2 Field Documentation	2050
6.213.2.1 filter_class_name	2050
6.213.2.2 filter_expression	2050
6.213.2.3 filter_parameters	2051
6.213.2.4 kind	2051
6.214 DDS_TransportBuiltinQosPolicy Struct Reference	2051
6.214.1 Detailed Description	2051
6.214.2 Field Documentation	2052
6.214.2.1 mask	2052
6.215 DDS_TransportInfo_t Struct Reference	2052
6.215.1 Detailed Description	2052
6.215.2 Field Documentation	2052
6.215.2.1 class_id	2053
6.215.2.2 message_size_max	2053
6.216 DDS_TransportInfoSeq Struct Reference	2053
6.216.1 Detailed Description	2053
6.217 DDS_TransportMulticastMapping_t Struct Reference	2053
6.217.1 Detailed Description	2054
6.217.2 Field Documentation	2054
6.217.2.1 addresses	2054
6.217.2.2 topic_expression	2055

6.217.2.3 mapping_function	2055
6.218 DDS_TransportMulticastMappingFunction_t Struct Reference	2055
6.218.1 Detailed Description	2055
6.218.2 Field Documentation	2055
6.218.2.1 dll	2056
6.218.2.2 function_name	2056
6.219 DDS_TransportMulticastMappingQosPolicy Struct Reference	2056
6.219.1 Detailed Description	2057
6.219.2 Field Documentation	2058
6.219.2.1 value	2058
6.220 DDS_TransportMulticastMappingSeq Struct Reference	2058
6.220.1 Detailed Description	2058
6.221 DDS_TransportMulticastQosPolicy Struct Reference	2058
6.221.1 Detailed Description	2059
6.221.2 Field Documentation	2059
6.221.2.1 value	2059
6.221.2.2 kind	2060
6.222 DDS_TransportMulticastSettings_t Struct Reference	2060
6.222.1 Detailed Description	2060
6.222.2 Field Documentation	2061
6.222.2.1 transports	2061
6.222.2.2 receive_address	2061
6.222.2.3 receive_port	2062
6.223 DDS_TransportMulticastSettingsSeq Struct Reference	2062
6.223.1 Detailed Description	2062
6.224 DDS_TransportPriorityQosPolicy Struct Reference	2062
6.224.1 Detailed Description	2063
6.224.2 Usage	2063
6.224.3 Field Documentation	2064
6.224.3.1 value	2064
6.225 DDS_TransportSelectionQosPolicy Struct Reference	2064
6.225.1 Detailed Description	2064
6.225.2 Field Documentation	2065
6.225.2.1 enabled_transports	2065
6.226 DDS_TransportUnicastQosPolicy Struct Reference	2065
6.226.1 Detailed Description	2066
6.226.2 Usage	2066
6.226.3 Field Documentation	2067
6.226.3.1 value	2067

6.227 DDS_TransportUnicastSettings_t Struct Reference	2067
6.227.1 Detailed Description	2068
6.227.2 Field Documentation	2068
6.227.2.1 transports	2068
6.227.2.2 receive_port	2068
6.228 DDS_TransportUnicastSettingsSeq Struct Reference	2069
6.228.1 Detailed Description	2069
6.229 DDS_TrustAlgorithmRequirements Struct Reference	2069
6.229.1 Detailed Description	2069
6.229.2 Field Documentation	2069
6.229.2.1 supported_mask	2069
6.229.2.2 required_mask	2070
6.230 DDS_TypeAllocationParams_t Struct Reference	2070
6.230.1 Detailed Description	2070
6.230.2 Field Documentation	2070
6.230.2.1 allocate_pointers	2070
6.230.2.2 allocate_optional_members	2071
6.231 DDS_TypeCode Struct Reference	2071
6.231.1 Detailed Description	2071
6.232 DDS_TypeCodeFactory Struct Reference	2071
6.232.1 Detailed Description	2072
6.233 DDS_TypeCodePrintFormatProperty Struct Reference	2073
6.233.1 Detailed Description	2073
6.233.2 Field Documentation	2073
6.233.2.1 indent	2073
6.233.2.2 print_ordinals	2074
6.233.2.3 print_kind	2074
6.233.2.4 print_complete_type	2075
6.234 DDS_TypeConsistencyEnforcementQosPolicy Struct Reference	2076
6.234.1 Detailed Description	2076
6.234.2 Field Documentation	2077
6.234.2.1 kind	2077
6.234.2.2 ignore_sequence_bounds	2078
6.234.2.3 ignore_string_bounds	2078
6.234.2.4 ignore_member_names	2078
6.234.2.5 prevent_type_widening	2079
6.234.2.6 force_type_validation	2079
6.234.2.7 ignore_enum_literal_names	2079
6.235 DDS_TypeDeallocationParams_t Struct Reference	2080

6.235.1 Detailed Description	2080
6.235.2 Field Documentation	2080
6.235.2.1 delete_pointers	2080
6.235.2.2 delete_optional_members	2081
6.236 DDS_TypeSupportQosPolicy Struct Reference	2081
6.236.1 Detailed Description	2081
6.236.2 Usage	2082
6.236.3 Field Documentation	2082
6.236.3.1 plugin_data	2082
6.236.3.2 cdr_padding_kind	2083
6.237 DDS_UInt8Seq Struct Reference	2083
6.237.1 Detailed Description	2083
6.238 DDS_UnionMember Struct Reference	2083
6.238.1 Detailed Description	2084
6.238.2 Field Documentation	2084
6.238.2.1 name	2084
6.238.2.2 is_pointer	2084
6.238.2.3 labels	2085
6.238.2.4 type	2085
6.239 DDS_UnionMemberSeq Struct Reference	2085
6.239.1 Detailed Description	2085
6.240 DDS_UnsignedLongLongSeq Struct Reference	2085
6.240.1 Detailed Description	2085
6.241 DDS_UnsignedLongSeq Struct Reference	2086
6.241.1 Detailed Description	2086
6.242 DDS_UnsignedShortSeq Struct Reference	2086
6.242.1 Detailed Description	2086
6.243 DDS_UserDataQosPolicy Struct Reference	2086
6.243.1 Detailed Description	2087
6.243.2 Usage	2087
6.243.3 Field Documentation	2087
6.243.3.1 value	2088
6.244 DDS_ValueMember Struct Reference	2088
6.244.1 Detailed Description	2088
6.244.2 Field Documentation	2089
6.244.2.1 name	2089
6.244.2.2 type	2089
6.244.2.3 is_pointer	2089
6.244.2.4 bits	2089

6.244.2.5 is_key	2089
6.244.2.6 access	2090
6.244.2.7 id	2090
6.244.2.8 is_optional	2090
6.245 DDS_ValueMemberSeq Struct Reference	2090
6.245.1 Detailed Description	2090
6.246 DDS_VendorId_t Struct Reference	2090
6.246.1 Detailed Description	2091
6.246.2 Field Documentation	2091
6.246.2.1 vendorId	2091
6.247 DDS_VirtualWriterInfo Struct Reference	2091
6.247.1 Detailed Description	2091
6.248 DDS_WaitSetProperty_t Struct Reference	2091
6.248.1 Detailed Description	2092
6.248.2 Field Documentation	2092
6.248.2.1 max_event_count	2092
6.248.2.2 max_event_delay	2093
6.249 DDS_WcharSeq Struct Reference	2093
6.249.1 Detailed Description	2093
6.250 DDS_WireProtocolQosPolicy Struct Reference	2093
6.250.1 Detailed Description	2094
6.250.2 Usage	2094
6.250.3 Field Documentation	2097
6.250.3.1 participant_id	2097
6.250.3.2 rtps_host_id	2098
6.250.3.3 rtps_app_id	2098
6.250.3.4 rtps_instance_id	2099
6.250.3.5 rtps_well_known_ports	2099
6.250.3.6 rtps_reserved_port_mask	2100
6.250.3.7 rtps_auto_id_kind	2100
6.250.3.8 compute_crc	2100
6.250.3.9 check_crc	2101
6.251 DDS_WriteParams_t Struct Reference	2101
6.251.1 Detailed Description	2102
6.251.2 Field Documentation	2102
6.251.2.1 replace_auto	2102
6.251.2.2 identity	2102
6.251.2.3 related_sample_identity	2103
6.251.2.4 source_timestamp	2103

6.251.2.5 cookie	2103
6.251.2.6 handle	2103
6.251.2.7 priority	2104
6.251.2.8 flag	2104
6.251.2.9 source_guid	2105
6.251.2.10 related_source_guid	2105
6.251.2.11 related_reader_guid	2106
6.252 DDS_WriterDataLifecycleQosPolicy Struct Reference	2106
6.252.1 Detailed Description	2106
6.252.2 Usage	2107
6.252.3 Field Documentation	2107
6.252.3.1 autodispose_unregistered_instances	2107
6.252.3.2 autopurge_unregistered_instances_delay	2108
6.252.3.3 autopurge_disposed_instances_delay	2108
6.253 DDS_WstringSeq Struct Reference	2109
6.253.1 Detailed Description	2109
6.254 Foo Struct Reference	2109
6.254.1 Detailed Description	2109
6.255 FooBarReplier Struct Reference	2110
6.255.1 Detailed Description	2110
6.256 FooBarRequester Struct Reference	2111
6.256.1 Detailed Description	2111
6.257 FooBarSimpleReplier Struct Reference	2112
6.257.1 Detailed Description	2112
6.258 FooDataReader Struct Reference	2112
6.258.1 Detailed Description	2112
6.259 FooDataWriter Struct Reference	2113
6.259.1 Detailed Description	2113
6.260 FooSeq Struct Reference	2113
6.260.1 Detailed Description	2113
6.261 FooTypeSupport Struct Reference	2114
6.261.1 Detailed Description	2114
6.262 NDDS_Config_LibraryVersion_t Struct Reference	2114
6.262.1 Detailed Description	2115
6.262.2 Field Documentation	2115
6.262.2.1 major	2115
6.262.2.2 minor	2115
6.262.2.3 release	2115
6.262.2.4 build	2116

6.263 NDDS_Config_Logger Struct Reference	2116
6.263.1 Detailed Description	2116
6.264 NDDS_Config_LoggerDevice Struct Reference	2116
6.264.1 Detailed Description	2116
6.264.2 Field Documentation	2117
6.264.2.1 write	2117
6.264.2.2 close	2117
6.265 NDDS_Config_LogMessage Struct Reference	2117
6.265.1 Detailed Description	2118
6.265.2 Field Documentation	2118
6.265.2.1 text	2118
6.265.2.2 level	2118
6.265.2.3 is_security_message	2118
6.265.2.4 is_license_message	2119
6.265.2.5 message_id	2119
6.265.2.6 timestamp	2119
6.265.2.7 facility	2119
6.266 NDDS_Config_Version_t Struct Reference	2119
6.266.1 Detailed Description	2119
6.267 NDDS_Transport_Address_t Struct Reference	2120
6.267.1 Detailed Description	2120
6.267.2 Field Documentation	2120
6.267.2.1 network_ordered_value	2120
6.268 NDDS_Transport_Interface_t Struct Reference	2120
6.268.1 Detailed Description	2121
6.268.2 Field Documentation	2121
6.268.2.1 transport_classid	2121
6.268.2.2 address	2121
6.268.2.3 status	2121
6.268.2.4 rank	2122
6.269 NDDS_Transport_Property_t Struct Reference	2122
6.269.1 Detailed Description	2123
6.269.2 Field Documentation	2123
6.269.2.1 classid	2123
6.269.2.2 address_bit_count	2124
6.269.2.3 properties_bitmap	2124
6.269.2.4 gather_send_buffer_count_max	2124
6.269.2.5 message_size_max	2125
6.269.2.6 allow_interfaces_list	2125

6.269.2.7 allow_interfaces_list_length	2126
6.269.2.8 deny_interfaces_list	2126
6.269.2.9 deny_interfaces_list_length	2126
6.269.2.10 allow_multicast_interfaces_list	2127
6.269.2.11 allow_multicast_interfaces_list_length	2127
6.269.2.12 deny_multicast_interfaces_list	2127
6.269.2.13 deny_multicast_interfaces_list_length	2128
6.269.2.14 transport_uuid	2128
6.269.2.15 thread_name_prefix	2128
6.269.2.16 max_interface_count	2129
6.270 NDDS_Transport_Shmem_Property_t Struct Reference	2129
6.270.1 Detailed Description	2130
6.270.2 Field Documentation	2130
6.270.2.1 parent	2130
6.270.2.2 received_message_count_max	2130
6.270.2.3 receive_buffer_size	2131
6.270.2.4 enable_udp_debugging	2131
6.270.2.5 udp_debugging_address	2132
6.270.2.6 udp_debugging_port	2132
6.271 NDDS_Transport_Support Struct Reference	2132
6.271.1 Detailed Description	2132
6.272 NDDS_Transport_UDP_WAN_CommPortsMappingInfo Struct Reference	2132
6.272.1 Detailed Description	2133
6.272.2 Field Documentation	2133
6.272.2.1 rtps_port	2133
6.272.2.2 host_port	2133
6.272.2.3 public_port	2133
6.273 NDDS_Transport_UDPv4_Property_t Struct Reference	2133
6.273.1 Detailed Description	2135
6.273.2 Field Documentation	2135
6.273.2.1 parent	2135
6.273.2.2 send_socket_buffer_size	2135
6.273.2.3 recv_socket_buffer_size	2136
6.273.2.4 unicast_enabled	2137
6.273.2.5 multicast_enabled	2137
6.273.2.6 multicast_ttl	2137
6.273.2.7 multicast_loopback_disabled	2137
6.273.2.8 ignore_loopback_interface	2138
6.273.2.9 ignore_nonup_interfaces	2138

6.273.2.10 ignore_nonrunning_interfaces	2139
6.273.2.11 no_zero_copy	2139
6.273.2.12 send_blocking	2139
6.273.2.13 use_checksum	2140
6.273.2.14 transport_priority_mask	2140
6.273.2.15 transport_priority_mapping_low	2140
6.273.2.16 transport_priority_mapping_high	2141
6.273.2.17 send_ping	2141
6.273.2.18 force_interface_poll_detection	2141
6.273.2.19 interface_poll_period	2141
6.273.2.20 reuse_multicast_receive_resource	2142
6.273.2.21 protocol_overhead_max	2142
6.273.2.22 disable_interface_tracking	2142
6.273.2.23 join_multicast_group_timeout	2143
6.273.2.24 public_address	2143
6.273.2.25 port_offset	2144
6.274 NDDS_Transport_UDPv4_WAN_Property_t Struct Reference	2144
6.274.1 Detailed Description	2145
6.274.2 Field Documentation	2145
6.274.2.1 parent	2145
6.274.2.2 send_socket_buffer_size	2146
6.274.2.3 recv_socket_buffer_size	2146
6.274.2.4 ignore_loopback_interface	2146
6.274.2.5 ignore_nonup_interfaces	2147
6.274.2.6 ignore_nonrunning_interfaces	2147
6.274.2.7 no_zero_copy	2148
6.274.2.8 send_blocking	2148
6.274.2.9 use_checksum	2148
6.274.2.10 transport_priority_mask	2149
6.274.2.11 transport_priority_mapping_low	2149
6.274.2.12 transport_priority_mapping_high	2149
6.274.2.13 send_ping	2150
6.274.2.14 force_interface_poll_detection	2150
6.274.2.15 interface_poll_period	2150
6.274.2.16 protocol_overhead_max	2150
6.274.2.17 disable_interface_tracking	2151
6.274.2.18 public_address	2151
6.274.2.19 comm_ports_list	2152
6.274.2.20 comm_ports_list_length	2153

6.274.2.21 port_offset	2153
6.274.2.22 binding_ping_period	2153
6.275 NDDS_Transport_UDPv6_Property_t Struct Reference	2154
6.275.1 Detailed Description	2155
6.275.2 Field Documentation	2155
6.275.2.1 parent	2155
6.275.2.2 send_socket_buffer_size	2156
6.275.2.3 recv_socket_buffer_size	2156
6.275.2.4 unicast_enabled	2157
6.275.2.5 multicast_enabled	2157
6.275.2.6 multicast_ttl	2157
6.275.2.7 multicast_loopback_disabled	2157
6.275.2.8 ignore_loopback_interface	2158
6.275.2.9 ignore_nonrunning_interfaces	2158
6.275.2.10 no_zero_copy	2159
6.275.2.11 send_blocking	2159
6.275.2.12 enable_v4mapped	2159
6.275.2.13 transport_priority_mask	2160
6.275.2.14 transport_priority_mapping_low	2160
6.275.2.15 transport_priority_mapping_high	2160
6.275.2.16 send_ping	2161
6.275.2.17 force_interface_poll_detection	2161
6.275.2.18 interface_poll_period	2161
6.275.2.19 reuse_multicast_receive_resource	2161
6.275.2.20 protocol_overhead_max	2162
6.275.2.21 disable_interface_tracking	2162
6.275.2.22 join_multicast_group_timeout	2162
6.275.2.23 public_address	2163
6.275.2.24 port_offset	2163
6.276 NDDS_Transport_UUID Struct Reference	2164
6.276.1 Detailed Description	2164
6.277 NDDS_Utility_HeapMonitoringParams_t Struct Reference	2164
6.277.1 Detailed Description	2164
6.277.2 Field Documentation	2164
6.277.2.1 snapshot_output_format	2164
6.277.2.2 snapshot_content_format	2165
6.278 NDDS_Utility_NetworkCaptureParams_t Struct Reference	2165
6.278.1 Detailed Description	2165
6.278.2 Field Documentation	2165

6.278.2.1 transports	2166
6.278.2.2 dropped_content	2166
6.278.2.3 traffic	2166
6.278.2.4 parse_encrypted_content	2166
6.278.2.5 checkpoint_thread_settings	2167
6.278.2.6 frame_queue_size	2167
6.279 RTI_Connext_Replier Struct Reference	2167
6.279.1 Detailed Description	2168
6.280 RTI_Connext_ReplierListener Struct Reference	2168
6.280.1 Detailed Description	2168
6.280.2 Field Documentation	2168
6.280.2.1 user_data	2169
6.281 RTI_Connext_ReplierParams Struct Reference	2169
6.281.1 Detailed Description	2169
6.281.2 Field Documentation	2170
6.281.2.1 participant	2170
6.281.2.2 service_name	2170
6.281.2.3 request_topic_name	2170
6.281.2.4 reply_topic_name	2170
6.281.2.5 qos_library_name	2171
6.281.2.6 qos_profile_name	2171
6.281.2.7 datawriter_qos	2171
6.281.2.8 datareader_qos	2172
6.281.2.9 publisher	2172
6.281.2.10 subscriber	2172
6.281.2.11 listener	2172
6.282 RTI_Connext_Requester Struct Reference	2173
6.282.1 Detailed Description	2173
6.283 RTI_Connext_RequesterParams Struct Reference	2173
6.283.1 Detailed Description	2174
6.283.2 Field Documentation	2174
6.283.2.1 participant	2174
6.283.2.2 service_name	2174
6.283.2.3 request_topic_name	2175
6.283.2.4 reply_topic_name	2175
6.283.2.5 qos_library_name	2175
6.283.2.6 qos_profile_name	2175
6.283.2.7 datawriter_qos	2176
6.283.2.8 datareader_qos	2176

6.283.2.9 publisher	2176
6.283.2.10 subscriber	2176
6.284 RTI_Connext_SimpleReplierListener Struct Reference	2176
6.284.1 Detailed Description	2177
6.284.2 Field Documentation	2177
6.284.2.1 user_data	2177
6.285 RTI_Connext_SimpleReplierParams Struct Reference	2177
6.285.1 Detailed Description	2178
6.285.2 Field Documentation	2178
6.285.2.1 participant	2178
6.285.2.2 service_name	2179
6.285.2.3 request_topic_name	2179
6.285.2.4 reply_topic_name	2179
6.285.2.5 qos_library_name	2179
6.285.2.6 qos_profile_name	2180
6.285.2.7 datawriter_qos	2180
6.285.2.8 datareader_qos	2180
6.285.2.9 publisher	2180
6.285.2.10 subscriber	2181
6.285.2.11 simple_listener	2181
6.286 TransportAllocationSettings_t Struct Reference	2181
6.286.1 Detailed Description	2181
7 File Documentation	2183
7.1 howto_prototypes.h	2183
7.2 transport_common.h	2184
7.3 transport_common_user.h	2189
7.4 transport_interface_user.h	2198
7.5 transport_shmem.h	2201
7.6 transport_udp.h	2205
7.7 transport_udp4.h	2217
7.8 transport_udp6.h	2229
8 Example Documentation	2237
8.1 HelloWorld.idl	2237
8.1.1 IDL Type Description	2237
8.1.1.1 HelloWorld.idl	2237
8.2 HelloWorld.c	2238
8.2.1 Programming Language Type Description	2238
8.2.1.1 HelloWorld.h	2238

8.2.1.2 HelloWorld.c	2239
8.3 HelloWorldSupport.c	2243
8.3.1 User Data Type Support	2243
8.3.1.1 HelloWorldSupport.h	2243
8.3.1.2 HelloWorldSupport.c	2244
8.4 HelloWorldPlugin.c	2245
8.4.1 RTI Connex Implementation Support	2245
8.4.1.1 HelloWorldPlugin.h	2245
8.4.1.2 HelloWorldPlugin.c	2247
8.5 HelloWorld_publisher.c	2256
8.5.1 RTI Connex Publication Example	2256
8.5.1.1 HelloWorld_publisher.c	2256
8.6 HelloWorld_subscriber.c	2259
8.6.1 RTI Connex Subscription Example	2259
8.6.1.1 HelloWorld_subscriber.c	2259
Index	2263

Chapter 1

RTI Connex

Core Libraries and Utilities

Real-Time Innovations, Inc.

RTI Connex 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 Connex uses the publish-subscribe communications model to make data distribution efficient and robust.

The RTI Connex 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".

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 Connex, the **Documentation Roadmap** (p. 1010) 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 Connex 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 rtdsgen to generate code from data types.
- **User's Manual.** Introduction to RTI Connex, product tour and conceptual presentation of the functionality of RTI Connex.
- **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 Connex DDS API Reference** (p. 1013) - The RTI Connex API reference.
- **Additional RTI Connex Communication Patterns** (p. 1017) - RTI Connex APIs for additional communication patterns
- **Programming How-To's** (p. 1018) - 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 Connex 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	1010
Conventions	1011
RTI Connext DDS API Reference	1013
Domain Module	26
DomainParticipantFactory	27
DomainParticipantConfigParams	1430
DomainParticipants	66
Built-in Topics	165
Participant Built-in Topics	1091
Topic Built-in Topics	1093
Publication Built-in Topics	1094
Subscription Built-in Topics	1096
ServiceRequest Built-in Topic	1098
Common types and functions	1102
Topic Module	167
Topics	168
FlatData Topic-Types	209
Zero Copy Transfer Over Shared Memory	210
User Data Type Support	210
Type Code Support	228
Built-in Types	305
String Built-in Type	1111
KeyedString Built-in Type	1123
Octets Built-in Type	1151
KeyedOctets Built-in Type	1169
Built-in Topic's Trust Types	309
Dynamic Data	313
DDS-Specific Primitive Types	1199
Publication Module	438
Publishers	439

Data Writers	471
Flow Controllers	556
Multi-channel DataWriters	719
Subscription Module	567
Subscribers	568
DataReaders	601
Read Conditions	694
Query Conditions	698
Topic Queries	703
Data Samples	700
Sample States	709
View States	711
Instance States	713
Stream Kinds	715
SampleProcessor	1503
Infrastructure Module	716
Clock Selection	25
Time Support	1205
GUID Support	1210
Sequence Number Support	1214
Exception Codes	1219
Return Codes	1220
Status Kinds	1222
QoS	1238
All QoS Policies	1238
ASYNCHRONOUS_PUBLISHER	1250
AVAILABILITY	1250
BATCH	1251
DATABASE	1252
DATA_READER_PROTOCOL	1253
DATA_READER_RESOURCE_LIMITS	1253
DATA_REPRESENTATION	1256
Compression Settings	1259
DATA_TAG	1263
DATA_WRITER_PROTOCOL	1268
DATA_WRITER_RESOURCE_LIMITS	1269
DATA_WRITER_TRANSFER_MODE	1272
DEADLINE	1272
DESTINATION_ORDER	1273
DISCOVERY	1275
NDDS_DISCOVERY_PEERS	1352
DISCOVERY_CONFIG	1276
DOMAIN_PARTICIPANT_RESOURCE_LIMITS	1283
DURABILITY	1285
DURABILITY_SERVICE	1289
ENTITY_FACTORY	1289
ENTITY_NAME	1290
EVENT	1291
HISTORY	1292
GROUP_DATA	1293
LATENCY_BUDGET	1294
LIFESPAN	1295
LIVELINESS	1296

LOCATORFILTER	1297
LOGGING	1298
MONITORING	1299
MULTICHANNEL	1300
OWNERSHIP	1301
OWNERSHIP_STRENGTH	1302
PARTITION	1303
PRESENTATION	1304
PROFILE	1305
PROPERTY	1306
PUBLISH_MODE	1317
READER_DATA_LIFECYCLE	1320
RECEIVER_POOL	1320
RELIABILITY	1321
RESOURCE_LIMITS	1325
SERVICE	1326
SYSTEM_RESOURCE_LIMITS	1328
TIME_BASED_FILTER	1329
TOPIC_DATA	1329
TOPIC_QUERY_DISPATCH	1330
TRANSPORT_BUILTIN	1331
TRANSPORT_MULTICAST	1335
Multicast Settings	1351
Multicast Mapping	1351
TRANSPORT_MULTICAST_MAPPING	1336
TRANSPORT_PRIORITY	1337
TRANSPORT_SELECTION	1338
TRANSPORT_UNICAST	1339
Unicast Settings	1350
TYPE_CONSISTENCY_ENFORCEMENT	1339
TYPESUPPORT	1341
USER_DATA	1343
WRITER_DATA_LIFECYCLE	1344
WIRE_PROTOCOL	1344
Extended Qos Support	1350
Thread Settings	1235
QoS by Category	1249
Basic QoS	783
DDS Specification QoS	790
DDS Extension QoS	795
Functional Categories	833
Application Acknowledgement	835
Application-Defined Metadata	836
Application Startup	838
Batching	838
Concurrency	839
Content Filtering	841
Data Availability	843
Debuggability	847
Discovery	853
Discovery Isolation	858
Durability	860
Flow Control	863
Fragmentation	866

Instance Lifecycle	867
Sending Large Data	871
Liveliness	874
Logging	875
Middleware Memory Management	878
QoS Profile Management	893
Reliability QoS	895
Sample Memory Management QoS	902
Sample Ordering	905
Security	906
Throughput	908
Transports	908
Topic Queries	911
Type System	912
QoS Mutability	914
Immutable QoS	914
Mutable QoS	949
Other QoS Attributes	956
Applied Per-Instance	956
Discovery Metadata	958
Matching	963
Deprecated QoS	965
Entity Support	1359
Conditions and WaitSets	1367
AsyncWaitSet	1067
Cookie	1385
Sample Flags	1386
WriteParams	1389
Heap Support in C	1392
Builtin Qos Profiles	1394
User-managed Threads	1432
Octet Buffer Support	1500
Sequence Support	1511
Built-in Sequences	718
String Support	1526
Transports	720
Installing Transport Plugins	725
Built-in Transport Plugins	736
UDP Transport Plugin definitions	1033
Shared Memory Transport	1035
UDIPv4 Transport	1041
Real-Time WAN Transport	1051
UDIPv6 Transport	1058
Creating New Transport Plugins	737
Transport Plugins Configuration	1020
Transport Address	1027
Queries and Filters Syntax	738
Logging and Version	744
Version	1436
Logging	1438
Activity Context	1458
General Utilities and Compliance Configuration	744
Compliance Configuration	1463

Heap Monitoring	.1477
Network Capture	.1483
Other Utilities	.1498
Observability Framework	.745
Monitoring Library 2.0	.1434
Durability and Persistence	.777
System Properties	.970
Configuring QoS Profiles with XML	.971
Additional RTI Connex Communication Patterns	.1017
Request-Reply Pattern	.745
Requester	.746
Replier	.761
Utilities	.776
Programming How-To's	.1018
Publication Example	.973
Subscription Example	.974
Participant Use Cases	.975
Topic Use Cases	.977
FlowController Use Cases	.978
Publisher Use Cases	.981
DataWriter Use Cases	.982
Subscriber Use Cases	.983
DataReader Use Cases	.985
Entity Use Cases	.988
Waitset Use Cases	.991
Transport Use Cases	.992
Filter Use Cases	.995
Creating Custom Content Filters	.998
Large Data Use Cases	.1001
Request-Reply Examples	.1002
Interface	.1019

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

DDS_AcknowledgmentInfo	
Information about an application-level acknowledged sample	1535
DDS_AckResponseData_t	
Data payload of an application-level acknowledgment	1536
DDS_AllocationSettings_t	
Resource allocation settings	1537
DDS_AnnotationParameterValue	
Annotation parameter value	1538
DDS_AsynchronousPublisherQosPolicy	
<<extension>> (p. 795) Configures the mechanism that sends user data in an external middle-ware thread	1539
DDS_AsyncWaitSetListener	
<<interface>> (p. 1013) Listener for receiving event notifications related to the thread pool of the DDS_AsyncWaitSet (p. 1070)	1543
DDS_AsyncWaitSetProperty_t	
Specifies the DDS_AsyncWaitSet (p. 1070) behavior	1544
DDS_AvailabilityQosPolicy	
<<extension>> (p. 795) Configures the availability of data	1547
DDS_BatchQosPolicy	
<<extension>> (p. 795) Used to configure batching of multiple samples into a single network packet in order to increase throughput for small samples	1552
DDS_BooleanSeq	
Instantiates FooSeq (p. 2113) < DDS_Boolean (p. 1204) >	1556
DDS_BuiltinTopicKey_t	
The key type of the built-in topic types	1556
DDS_BuiltinTopicReaderResourceLimits_t	
<<extension>> (p. 795) Built-in topic reader's resource limits	1557
DDS_ChannelSettings_t	
<<extension>> (p. 795) Type used to configure the properties of a channel	1564
DDS_ChannelSettingsSeq	
Declares IDL <i>sequence</i> < DDS_ChannelSettings_t (p. 1564) >	1566

DDS_CharSeq	
Instantiates FooSeq (p. 2113) < DDS_Char (p. 1201) >	1566
DDS_CoherentSetInfo_t	
<<extension>> (p. 1012) Type definition for a coherent set info	1566
DDS_CompressionSettings_t	
<<extension>> (p. 795) Settings related to compressing user data	1568
DDS_ConditionHandler	
<<extension>> (p. 1012) <<interface>> (p. 1013) Handler called by the DDS_Condition ↔	
dispatch (p. 1375)	1570
DDS_ConditionSeq	
Instantiates FooSeq (p. 2113) < DDS_Condition (p. 1370) >	1571
DDS_ContentFilter	
<<interface>> (p. 1013) Interface to be used by a custom filter of a DDS_ContentFilteredTopic	
(p. 177)	1571
DDS_ContentFilterProperty_t	
<<extension>> (p. 1012) Type used to provide all the required information to enable content filter-	
ing	1579
DDS_Cookie_t	
<<extension>> (p. 1012) Sequence of bytes	1580
DDS_CookieSeq	
Declares IDL sequence < DDS_Cookie_t (p. 1580) >	1581
DDS_DatabaseQosPolicy	
<<extension>> (p. 795) Various threads and resource limits settings used by RTI Connext to	
control its internal database	1581
DDS_DataReaderCacheStatus	
<<extension>> (p. 1012) The status of the reader's cache	1586
DDS_DataReaderListener	
<<interface>> (p. 1013) DDS_Listener (p. 1816) for reader status	1593
DDS_DataReaderProtocolQosPolicy	
<<extension>> (p. 795) Along with DDS_WireProtocolQosPolicy (p. 2093) and DDS_Data ↔	
WriterProtocolQosPolicy (p. 1649), this QoS policy configures the DDS on-the-network protocol	
(RTPS)	1596
DDS_DataReaderProtocolStatus	
<<extension>> (p. 1012) The status of a reader's internal protocol related metrics, like the number	
of samples received, filtered, rejected; and status of wire protocol traffic	1600
DDS_DataReaderQos	
QoS policies supported by a DDS_DataReader (p. 615) entity	1611
DDS_DataReaderResourceLimitsInstanceReplacementSettings	
Instance replacement kind applied to each instance state	1618
DDS_DataReaderResourceLimitsQosPolicy	
<<extension>> (p. 795) Various settings that configure how a DDS_DataReader (p. 615) allocates	
and uses physical memory for internal resources	1619
DDS_DataReaderSeq	
Declares IDL sequence < DDS_DataReader (p. 615) >	1637
DDS_DataRepresentationIdSeq	
Declares IDL sequence < DDS_DataRepresentationId_t (p. 1257) >	1638
DDS_DataRepresentationQosPolicy	
<<spec>> (p. 790) This QoS policy contains a list of representation identifiers and compression	
settings used by DDS_DataWriter (p. 485) and DDS_DataReader (p. 615) entities to negotiate which	
data representation and compression settings to use	1638
DDS_DataTags	
Definition of DDS_DataTagQosPolicy (p. 1264)	1641

DDS_DataWriterCacheStatus	
<<extension>> (p. 1012) 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	1641
DDS_DataWriterListener	
<<interface>> (p. 1013) DDS_Listener (p. 1816) for writer status	1644
DDS_DataWriterProtocolQosPolicy	
<<extension>> (p. 795) Protocol that applies only to DDS_DataWriter (p. 485) instances	1649
DDS_DataWriterProtocolStatus	
<<extension>> (p. 1012) The status of a writer's internal protocol related metrics, like the number of samples pushed, pulled, filtered; and status of wire protocol traffic	1654
DDS_DataWriterQos	
QoS policies supported by a DDS_DataWriter (p. 485) entity	1666
DDS_DataWriterResourceLimitsQosPolicy	
<<extension>> (p. 795) Various settings that configure how a DDS_DataWriter (p. 485) allocates and uses physical memory for internal resources	1673
DDS_DataWriterShmemRefTransferModeSettings	
Settings related to transferring data using shared memory references	1682
DDS_DataWriterTransferModeQosPolicy	
<<extension>> (p. 795) Qos related to transferring data	1683
DDS_DeadlineQosPolicy	
<<spec>> (p. 790) Expresses the maximum duration (deadline) within which an instance is expected to be updated	1684
DDS_DestinationOrderQosPolicy	
<<spec>> (p. 790) Controls how the middleware will deal with data sent by multiple DDS_DataWriter (p. 485) entities for the same instance of data (i.e., same DDS_Topic (p. 175) and key)	1686
DDS_DiscoveryConfigQosPolicy	
Settings for discovery configuration	1689
DDS_DiscoveryQosPolicy	
<<extension>> (p. 795) Configures the mechanism used by the middleware to automatically discover and connect with new remote applications	1715
DDS_DomainParticipantConfigParams_t	
<<extension>> (p. 1012) 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	1719
DDS_DomainParticipantFactoryQos	
QoS policies supported by a DDS_DomainParticipantFactory (p. 32)	1721
DDS_DomainParticipantListener	
<<interface>> (p. 1013) Listener for participant status	1723
DDS_DomainParticipantProtocolStatus	
<<extension>> (p. 1012) 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	1725
DDS_DomainParticipantQos	
QoS policies supported by a DDS_DomainParticipant (p. 77) entity	1726
DDS_DomainParticipantResourceLimitsQosPolicy	
<<extension>> (p. 795) Various settings that configure how a DDS_DomainParticipant (p. 77) allocates and uses physical memory for internal resources, including the maximum sizes of various properties	1731
DDS_DomainParticipantSeq	
Declares IDL <i>sequence</i> < DDS_DomainParticipant (p. 77) >	1759
DDS_DoubleSeq	
Instantiates FooSeq (p. 2113) < DDS_Double (p. 1203) >	1759
DDS_DurabilityQosPolicy	
<<spec>> (p. 790) This QoS policy specifies whether or not RTI Connexet will store and deliver previously published data samples to new DDS_DataReader (p. 615) entities that join the network later	1759

DDS_DurabilityServiceQosPolicy	
<< <i>spec</i> >> (p. 790) Various settings to configure the external <i>RTI Persistence Service</i> used by RTI Connex for DataWriters with a DDS_DurabilityQosPolicy (p. 1759) setting of DDS_PERSISTENT ↔ _DURABILITY_QOS (p. 1288) or DDS_TRANSIENT_DURABILITY_QOS (p. 1288)	1764
DDS_Duration_t	
Type for <i>duration</i> representation	1767
DDS_DynamicData	
A sample of any complex data type, which can be inspected and manipulated reflectively	1768
DDS_DynamicDataInfo	
A descriptor for a DDS_DynamicData (p. 1768) object	1775
DDS_DynamicDataJsonParserProperties_t	
A collection of attributes used to configure DDS_DynamicData (p. 1768) objects	1776
DDS_DynamicDataMemberInfo	
A descriptor for a single member (i.e. field) of dynamically defined data type	1777
DDS_DynamicDataProperty_t	
A collection of attributes used to configure DDS_DynamicData (p. 1768) objects	1779
DDS_DynamicDataSeq	
An ordered collection of DDS_DynamicData (p. 1768) elements	1781
DDS_DynamicDataTypeProperty_t	
A collection of attributes used to configure DDS_DynamicDataTypeSupport (p. 326) objects	1781
DDS_DynamicDataTypeSerializationProperty_t	
Properties that govern how data of a certain type will be serialized on the network	1782
DDS_EndpointGroup_t	
Specifies a group of endpoints that can be collectively identified by a name and satisfied by a quorum 1784	
DDS_EndpointGroupSeq	
A sequence of DDS_EndpointGroup_t (p. 1784)	1785
DDS_EndpointTrustAlgorithmInfo	
Trust Plugins algorithm information associated with the discovered endpoint	1786
DDS_EndpointTrustInterceptorAlgorithmInfo	
Trust Plugins interception algorithm information associated with the discovered endpoint	1786
DDS_EndpointTrustProtectionInfo	
Trust Plugins Protection information associated with the discovered endpoint	1787
DDS_EntityFactoryQosPolicy	
<< <i>spec</i> >> (p. 790) A QoS policy for all DDS_Entity (p. 1360) types that can act as factories for one or more other DDS_Entity (p. 1360) types	1788
DDS_EntityNameQosPolicy	
<< <i>extension</i> >> (p. 795) Assigns a name and a role name to a DDS_DomainParticipant (p. 77), DDS_Publisher (p. 443), DDS_Subscriber (p. 572), DDS_DataWriter (p. 485) or DDS_DataReader (p. 615). Except for DDS_Publisher (p. 443) and DDS_Subscriber (p. 572), these names will be visible during the discovery process and in RTI tools to help you visualize and debug your system	1790
DDS_EnumMember	
A description of a member of an enumeration	1791
DDS_EnumMemberSeq	
Defines a sequence of enumerator members	1792
DDS_EventQosPolicy	
<< <i>extension</i> >> (p. 795) Settings for event	1793
DDS_ExpressionProperty	
Provides additional information about the filter expression passed to DDS_ContentFilter::writer ↔ compile (p. 1573)	1795
DDS_FilterSampleInfo	
Provides meta information associated with the sample	1796
DDS_FloatSeq	
Instantiates FooSeq (p. 2113) < DDS_Float (p. 1203) >	1798

DDS_FlowControllerProperty_t	
Determines the flow control characteristics of the DDS_FlowController (p. 559)	1799
DDS_FlowControllerTokenBucketProperty_t	
DDS_FlowController (p. 559) uses the popular token bucket approach for open loop network flow control. The flow control characteristics are determined by the token bucket properties	1800
DDS_GroupDataQosPolicy	
Attaches a buffer of opaque data that is distributed by means of Built-in Topics (p. 165) during discovery	1803
DDS_GUID_t	
Type for <i>GUID</i> (Global Unique Identifier) representation	1804
DDS_HistoryQosPolicy	
<< <i>spec</i> >> (p. 790) Specifies the behavior of RTI Connex 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	1805
DDS_InconsistentTopicStatus	
DDS_INCONSISTENT_TOPIC_STATUS (p. 1228)	1808
DDS_InstanceHandleSeq	
Instantiates FooSeq (p. 2113) < DDS_InstanceHandle_t (p. 214) >	1809
DDS_Int8Seq	
Instantiates FooSeq (p. 2113) < DDS_Int8 (p. 1202) >	1810
DDS_InvalidLocalAdvanceNoticeStatus	
DDS_INVALID_LOCAL_ADVANCE_NOTICE_STATUS (p. 1232)	1810
DDS_KeyedOctets	
Built-in type consisting of a variable-length array of opaque bytes and a string that is the key	1811
DDS_KeyedOctetsSeq	
Instantiates FooSeq (p. 2113) < DDS_KeyedOctets (p. 1811) >	1812
DDS_KeyedOctetsTypeSupport	
<< <i>interface</i> >> (p. 1013) DDS_KeyedOctets (p. 1811) type support	1812
DDS_KeyedString	
Keyed string built-in type	1812
DDS_KeyedStringSeq	
Instantiates FooSeq (p. 2113) < DDS_KeyedString (p. 1812) >	1813
DDS_KeyedStringTypeSupport	
<< <i>interface</i> >> (p. 1013) Keyed string type support	1813
DDS_LatencyBudgetQosPolicy	
<< <i>spec</i> >> (p. 790) 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	1813
DDS_LifespanQosPolicy	
<< <i>spec</i> >> (p. 790) Specifies how long the data written by the DDS_DataWriter (p. 485) is considered valid	1815
DDS_Listener	
<< <i>interface</i> >> (p. 1013) Abstract base class for all Listener interfaces	1816
DDS_LivelinessChangedStatus	
DDS_LIVELINESS_CHANGED_STATUS (p. 1231)	1820
DDS_LivelinessLostStatus	
DDS_LIVELINESS_LOST_STATUS (p. 1231)	1822
DDS_LivelinessQosPolicy	
Specifies and configures the mechanism that allows DDS_DataReader (p. 615) entities to detect when DDS_DataWriter (p. 485) entities become disconnected or "dead."	1823
DDS_Locator_t	
<< <i>extension</i> >> (p. 795) Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports	1826

DDS_LocatorFilter_t	
<<extension>> (p. 795) Specifies the configuration of an individual channel within a MultiChannel DataWriter	1828
DDS_LocatorFilterQosPolicy	
<<extension>> (p. 795) The QoS policy used to report the configuration of a MultiChannel DataWriter as part of DDS_PublicationBuiltinTopicData (p. 1905)	1829
DDS_LocatorFilterSeq	
Declares IDL sequence< DDS_LocatorFilter_t (p. 1828) >	1830
DDS_LocatorSeq	
Declares IDL sequence < DDS_Locator_t (p. 1826) >	1830
DDS_LoggingQosPolicy	
<<extension>> (p. 795) Configures the RTI Connex logging facility	1831
DDS_LongDoubleSeq	
Instantiates FooSeq (p. 2113) < DDS_LongDouble (p. 1204) >	1835
DDS_LongLongSeq	
Instantiates FooSeq (p. 2113) < DDS_LongLong (p. 1203) >	1835
DDS_LongSeq	
Instantiates FooSeq (p. 2113) < DDS_Long (p. 1203) >	1835
DDS_MonitoringDedicatedParticipantSettings	
<<extension>> (p. 795) Configures the use of a dedicated DDS_DomainParticipant (p. 77) to distribute the RTI Connex application telemetry data	1836
DDS_MonitoringDistributionSettings	
<<extension>> (p. 795) Configures the distribution of telemetry data	1838
DDS_MonitoringEventDistributionSettings	
<<extension>> (p. 795) Configures the distribution of event metrics	1840
DDS_MonitoringLoggingDistributionSettings	
<<extension>> (p. 795) Configures the distribution of log messages	1843
DDS_MonitoringLoggingForwardingSettings	
<<extension>> (p. 795) Configures the forwarding levels of log messages for the different NDDS_ _Config_LogFacility (p. 1446)	1847
DDS_MonitoringMetricSelection	
<<extension>> (p. 795) This data structure is used to configure event and periodic metrics collection and distribution for a specific set of observable resources	1849
DDS_MonitoringMetricSelectionSeq	
Declares IDL sequence < DDS_MonitoringMetricSelection (p. 1849) >	1851
DDS_MonitoringPeriodicDistributionSettings	
<<extension>> (p. 795) Configures the distribution of periodic metrics	1852
DDS_MonitoringQosPolicy	
<<extension>> (p. 795) Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connex telemetry data	1853
DDS_MonitoringTelemetryData	
<<extension>> (p. 795) Configures the telemetry data that will be distributed	1855
DDS_MultiChannelQosPolicy	
<<extension>> (p. 795) Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data	1856
DDS_Octets	
Built-in type consisting of a variable-length array of opaque bytes	1859
DDS_OctetSeq	
Instantiates FooSeq (p. 2113) < DDS_Octet (p. 1202) >	1860
DDS_OctetsSeq	
Instantiates FooSeq (p. 2113) < DDS_Octets (p. 1859) >	1860
DDS_OctetsTypeSupport	
<<interface>> (p. 1013) DDS_Octets (p. 1859) type support	1860

DDS_OfferedDeadlineMissedStatus	
DDS_OFFERED_DEADLINE_MISSED_STATUS (p. 1228)	1861
DDS_OfferedIncompatibleQosStatus	
DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 1229)	1862
DDS_OwnershipQosPolicy	
<< <i>spec</i> >> (p. 790) Specifies whether it is allowed for multiple DDS_DataWriter (p. 485) (s) to write the same instance of the data and if so, how these modifications should be arbitrated	1863
DDS_OwnershipStrengthQosPolicy	
Specifies the value of the strength used to arbitrate among multiple DDS_DataWriter (p. 485) objects that attempt to modify the same instance of a data type (identified by DDS_Topic (p. 175) + key)	1868
DDS_ParticipantBuiltinTopicData	
Entry created when a DomainParticipant object is discovered	1870
DDS_ParticipantBuiltinTopicDataSeq	
Instantiates FooSeq (p. 2113) < DDS_ParticipantBuiltinTopicData (p. 1870) >	1876
DDS_ParticipantBuiltinTopicDataTypeSupport	
Instantiates TypeSupport < DDS_ParticipantBuiltinTopicData (p. 1870) >	1876
DDS_ParticipantTrustAlgorithmInfo	
Trust Plugins algorithm information associated with the discovered DomainParticipant	1876
DDS_ParticipantTrustInterceptorAlgorithmInfo	
Trust Plugins interception algorithm information associated with the discovered DomainParticipant	1877
DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo	
Trust Plugins key establishment algorithm information associated with the discovered DomainParticipant	1879
DDS_ParticipantTrustProtectionInfo	
Trust Plugins Protection information associated with the discovered DomainParticipant	1879
DDS_ParticipantTrustSignatureAlgorithmInfo	
Trust Plugins signature algorithm information associated with the discovered DomainParticipant	1880
DDS_PartitionQosPolicy	
<< <i>spec</i> >> (p. 790) Set of strings that introduces logical partitions in DDS_DomainParticipant (p. 77), DDS_Publisher (p. 443), or DDS_Subscriber (p. 572) entities	1881
DDS_PersistentStorageSettings	
Configures durable writer history and durable reader state	1883
DDS_PresentationQosPolicy	
<< <i>spec</i> >> (p. 790) Specifies how the samples representing changes to data instances are presented to a subscribing application	1890
DDS_PrintFormatProperty	
A collection of attributes used to configure how data samples will be formatted when converted to a string	1894
DDS_ProductVersion_t	
<< <i>extension</i> >> (p. 1012) Type used to represent the current version of RTI Connext	1897
DDS_ProfileQosPolicy	
<< <i>extension</i> >> (p. 795) Configures the way that XML documents containing QoS profiles are loaded by RTI Connext	1898
DDS_Property_t	
Properties are name/value pairs objects	1900
DDS_PropertyQosPolicy	
<< <i>extension</i> >> (p. 795) 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	1901
DDS_PropertySeq	
Declares IDL <i>sequence</i> < DDS_Property_t (p. 1900) >	1903
DDS_ProtocolVersion_t	
<< <i>extension</i> >> (p. 1012) Type used to represent the version of the RTPS protocol	1904

DDS_PublicationBuiltinTopicData	
Entry created when a DDS_DataWriter (p. 485) is discovered in association with its Publisher . . .	1905
DDS_PublicationBuiltinTopicDataSeq	
Instantiates FooSeq (p. 2113) < DDS_PublicationBuiltinTopicData (p. 1905) >	1914
DDS_PublicationBuiltinTopicDataTypeSupport	
Instantiates TypeSupport < DDS_PublicationBuiltinTopicData (p. 1905) >	1914
DDS_PublicationMatchedStatus	
DDS_PUBLICATION_MATCHED_STATUS (p. 1232)	1914
DDS_PublisherListener	
<< <i>interface</i> >> (p. 1013) DDS_Listener (p. 1816) for DDS_Publisher (p. 443) status	1917
DDS_PublisherQos	
QoS policies supported by a DDS_Publisher (p. 443) entity	1918
DDS_PublisherSeq	
Declares IDL <i>sequence</i> < DDS_Publisher (p. 443) >	1920
DDS_PublishModeQosPolicy	
<< <i>extension</i> >> (p. 795) Specifies how RTI Connexx sends application data on the network. This QoS policy can be used to tell RTI Connexx to use its <i>own</i> thread to send data, instead of the user thread	1920
DDS_QosPolicyCount	
Type to hold a counter for a DDS_QosPolicyId_t (p. 1246)	1924
DDS_QosPolicyCountSeq	
Declares IDL <i>sequence</i> < DDS_QosPolicyCount (p. 1924) >	1925
DDS_QosPrintFormat	
A collection of attributes used to configure how a QoS appears when printed	1926
DDS_QueryConditionParams	
<< <i>extension</i> >> (p. 1012) Input parameters for DDS_DataReader_create_querycondition_w ↔ _params (p. 673)	1928
DDS_ReadConditionParams	
<< <i>extension</i> >> (p. 1012) Input parameters for DDS_DataReader_create_readcondition_w ↔ _params (p. 672)	1929
DDS_ReaderDataLifecycleQosPolicy	
<< <i>spec</i> >> (p. 790) Controls how a DataReader manages the lifecycle of the data that it has received	1930
DDS_ReceiverPoolQosPolicy	
<< <i>extension</i> >> (p. 795) Configures threads used by RTI Connexx to receive and process data from transports (for example, UDP sockets)	1934
DDS_ReliabilityQosPolicy	
<< <i>extension</i> >> (p. 795) << <i>spec</i> >> (p. 790) Indicates the level of reliability offered/requested by RTI Connexx	1936
DDS_ReliableReaderActivityChangedStatus	
<< <i>extension</i> >> (p. 1012) Describes the activity (i.e. are acknowledgements forthcoming) of reliable readers matched to a reliable writer	1940
DDS_ReliableWriterCacheChangedStatus	
<< <i>extension</i> >> (p. 1012) A summary of the state of a data writer's cache of unacknowledged samples written	1942
DDS_ReliableWriterCacheEventCount	
<< <i>extension</i> >> (p. 1012) The number of times the number of unacknowledged samples in the cache of a reliable writer hit a certain well-defined threshold	1945
DDS_RequestedDeadlineMissedStatus	
DDS_REQUESTED_DEADLINE_MISSED_STATUS (p. 1229)	1946
DDS_RequestedIncompatibleQosStatus	
DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS (p. 1229)	1947

DDS_ResourceLimitsQosPolicy	
<< <i>spec</i> >> (p. 790) 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	1949
DDS_RTPS_EntityId_t	
From the DDS-RTPS specification: type used to hold the suffix part of the globally-unique RTPS-entity identifiers	1953
DDS_RTPS_GUID_t	
From the DDS-RTPS specification: type used to hold a globally-unique RTPS-entity identifier	1954
DDS_RtpsReliableReaderProtocol_t	
<< <i>extension</i> >> (p. 795) QoS related to reliable reader protocol defined in RTPS	1954
DDS_RtpsReliableWriterProtocol_t	
<< <i>extension</i> >> (p. 795) QoS related to the reliable writer protocol defined in RTPS	1960
DDS_RtpsWellKnownPorts_t	
<< <i>extension</i> >> (p. 795) RTPS well-known port mapping configuration	1978
DDS_SampleHandler	
<< <i>experimental</i> >> (p. 1012) << <i>extension</i> >> (p. 1012) << <i>interface</i> >> (p. 1013) Handler called by a sample dispatcher, such as DDS_SampleProcessor (p. 1505)	1983
DDS_SampleIdentity_t	
Type definition for a Sample Identity	1984
DDS_SampleInfo	
Information that accompanies each sample that is read or taken	1985
DDS_SampleInfoSeq	
Declares IDL <i>sequence</i> < DDS_SampleInfo (p. 1985) >	1996
DDS_SampleLostStatus	
DDS_SAMPLE_LOST_STATUS (p. 1230)	1996
DDS_SampleRejectedStatus	
DDS_SAMPLE_REJECTED_STATUS (p. 1230)	1997
DDS_SequenceNumber_t	
Type for <i>sequence</i> number representation	1999
DDS_ServiceQosPolicy	
<< <i>extension</i> >> (p. 795) Service or tool associated with a DDS entity	2000
DDS_ServiceRequest	
A request coming from one of the built-in services	2001
DDS_ServiceRequestAcceptedStatus	
DDS_SERVICE_REQUEST_ACCEPTED_STATUS (p. 1233)	2002
DDS_ServiceRequestSeq	
Instantiates FooSeq (p. 2113) < DDS_ServiceRequest (p. 2001) >	2004
DDS_ServiceRequestTypeSupport	
Instantiates <i>TypeSupport</i> < DDS_ServiceRequest (p. 2001) >	2004
DDS_ShortSeq	
Instantiates FooSeq (p. 2113) < DDS_Short (p. 1202) >	2005
DDS_StringSeq	
Instantiates FooSeq (p. 2113) < <i>char*</i> > with value type semantics	2005
DDS_StringTypeSupport	
<< <i>interface</i> >> (p. 1013) String type support	2006
DDS_StructMember	
A description of a member of a struct	2007
DDS_StructMemberSeq	
Defines a sequence of struct members	2009
DDS_SubscriberListener	
<< <i>interface</i> >> (p. 1013) DDS_Listener (p. 1816) for status about a subscriber	2009
DDS_SubscriberQos	
QoS policies supported by a DDS_Subscriber (p. 572) entity	2010

DDS_SubscriberSeq	
Declares IDL <i>sequence</i> < DDS_Subscriber (p. 572) >	2012
DDS_SubscriptionBuiltinTopicData	
Entry created when a DDS_DataReader (p. 615) is discovered in association with its Subscriber	2012
DDS_SubscriptionBuiltinTopicDataSeq	
Instantiates FooSeq (p. 2113) < DDS_SubscriptionBuiltinTopicData (p. 2012) >	2021
DDS_SubscriptionBuiltinTopicDataTypeSupport	
Instantiates <i>TypeSupport</i> < DDS_SubscriptionBuiltinTopicData (p. 2012) >	2021
DDS_SubscriptionMatchedStatus	
DDS_SUBSCRIPTION_MATCHED_STATUS (p. 1232)	2022
DDS_SystemResourceLimitsQosPolicy	
<< <i>extension</i> >> (p. 795) Configures DDS_DomainParticipant (p. 77)-independent resources used by RTI Connext. Mainly used to change the maximum number of DDS_DomainParticipant (p. 77) entities that can be created within a single process (address space)	2024
DDS_Tag	
Tags are name/value pair objects	2026
DDS_TagSeq	
Declares IDL <i>sequence</i> < DDS_Tag (p. 2026) >	2027
DDS_ThreadFactory	
<< <i>extension</i> >> (p. 1012) << <i>interface</i> >> (p. 1013) Interface for providing the threads needed by the middleware. It consists of operations to create and delete threads. The interface depends on the DDS_ThreadFactory_OnSpawnedFunction (p. 1433) that specifies the operation to run in the new thread	2027
DDS_ThreadSettings_t	
The properties of a thread of execution	2029
DDS_Time_t	
Type for <i>time</i> representation	2031
DDS_TimeBasedFilterQosPolicy	
<< <i>spec</i> >> (p. 790) Filter that allows a DDS_DataReader (p. 615) to specify that it is interested only in (potentially) a subset of the values of the data	2032
DDS_TopicBuiltinTopicData	
Entry created when a Topic object discovered	2034
DDS_TopicBuiltinTopicDataSeq	
Instantiates FooSeq (p. 2113) < DDS_TopicBuiltinTopicData (p. 2034) >	2039
DDS_TopicBuiltinTopicDataTypeSupport	
Instantiates <i>TypeSupport</i> < DDS_TopicBuiltinTopicData (p. 2034) >	2039
DDS_TopicDataQosPolicy	
Attaches a buffer of opaque data that is distributed by means of Built-in Topics (p. 165) during discovery	2040
DDS_TopicListener	
<< <i>interface</i> >> (p. 1013) DDS_Listener (p. 1816) for DDS_Topic (p. 175) entities	2041
DDS_TopicQos	
QoS policies supported by a DDS_Topic (p. 175) entity	2042
DDS_TopicQueryData	
<< <i>extension</i> >> (p. 1012) Provides information about a DDS_TopicQuery (p. 707)	2046
DDS_TopicQueryDispatchQosPolicy	
<< <i>extension</i> >> (p. 795) Configures the ability of a DDS_DataWriter (p. 485) to publish samples in response to a DDS_TopicQuery (p. 707)	2047
DDS_TopicQuerySelection	
<< <i>extension</i> >> (p. 1012) Specifies the data query that defines a DDS_TopicQuery (p. 707)	2049
DDS_TransportBuiltinQosPolicy	
<< <i>extension</i> >> (p. 795) Specifies which built-in transports are used	2051
DDS_TransportInfo_t	
Contains the <i>class_id</i> and <i>message_size_max</i> of an installed transport	2052

DDS_TransportInfoSeq	Instantiates FooSeq (p. 2113) < DDS_TransportInfo_t (p. 2052) >	2053
DDS_TransportMulticastMapping_t	<< <i>extension</i> >> (p. 795) Type representing a list of multicast mapping elements	2053
DDS_TransportMulticastMappingFunction_t	Type representing an external mapping function	2055
DDS_TransportMulticastMappingQosPolicy	<< <i>extension</i> >> (p. 795) Specifies a list of topic_expressions and multicast addresses that can be used by an Entity with a specific topic name to receive data	2056
DDS_TransportMulticastMappingSeq	Declares IDL <i>sequence</i> < DDS_TransportMulticastMapping_t (p. 2053) >	2058
DDS_TransportMulticastQosPolicy	<< <i>extension</i> >> (p. 795) Specifies the multicast address on which a DDS_DataReader (p. 615) wants to receive its data. It can also specify a port number as well as a subset of the available (at the DDS_DomainParticipant (p. 77) level) transports with which to receive the multicast data	2058
DDS_TransportMulticastSettings_t	Type representing a list of multicast locators	2060
DDS_TransportMulticastSettingsSeq	Declares IDL <i>sequence</i> < DDS_TransportMulticastSettings_t (p. 2060) >	2062
DDS_TransportPriorityQosPolicy	<< <i>spec</i> >> (p. 790) This QoS policy allows the application to take advantage of transports that are capable of sending messages with different priorities	2062
DDS_TransportSelectionQosPolicy	<< <i>extension</i> >> (p. 795) Specifies the physical transports a DDS_DataWriter (p. 485) or DDS_← DataReader (p. 615) may use to send or receive data	2064
DDS_TransportUnicastQosPolicy	<< <i>extension</i> >> (p. 795) Specifies a subset of transports and a port number that can be used by an Entity to receive data	2065
DDS_TransportUnicastSettings_t	Type representing a list of unicast locators	2067
DDS_TransportUnicastSettingsSeq	Declares IDL <i>sequence</i> < DDS_TransportUnicastSettings_t (p. 2067) >	2069
DDS_TrustAlgorithmRequirements	Type to describe Trust Plugins algorithm requirements for an entity	2069
DDS_TypeAllocationParams_t	Configures whether or not to allocate pointer and optional members	2070
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 <code>rtiddsgen</code> (see the <code>Code Generator User's Manual</code>) or to modify types you define yourself at runtime	2071
DDS_TypeCodeFactory	A singleton factory for creating, copying, and deleting data type definitions dynamically	2071
DDS_TypeCodePrintFormatProperty	A collection of attributes used to configure how a <code>TypeCode</code> appears when converted to a string	2073
DDS_TypeConsistencyEnforcementQosPolicy	<< <i>spec</i> >> (p. 790) Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it	2076
DDS_TypeDeallocationParams_t	Configures whether to release or not pointer and optional members	2080
DDS_TypeSupportQosPolicy	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Allows you to attach application-specific values to a DDS_DataWriter (p. 485) or DDS_DataReader (p. 615), 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	2081

DDS_UInt8Seq	Instantiates FooSeq (p. 2113) < DDS_UInt8 (p. 1202) >	2083
DDS_UnionMember	A description of a member of a union	2083
DDS_UnionMemberSeq	Defines a sequence of union members	2085
DDS_UnsignedLongLongSeq	Instantiates FooSeq (p. 2113) < DDS_UnsignedLongLong (p. 1203) >	2085
DDS_UnsignedLongSeq	Instantiates FooSeq (p. 2113) < DDS_UnsignedLong (p. 1203) >	2086
DDS_UnsignedShortSeq	Instantiates FooSeq (p. 2113) < DDS_UnsignedShort (p. 1202) >	2086
DDS_UserDataQosPolicy	Attaches a buffer of opaque data that is distributed by means of Built-in Topics (p. 165) during discovery	2086
DDS_ValueMember	A description of a member of a value type	2088
DDS_ValueMemberSeq	Defines a sequence of value members	2090
DDS_VendorId_t	<< <i>extension</i> >> (p. 1012) Type used to represent the vendor of the service implementing the RTPS protocol	2090
DDS_VirtualWriterInfo	The structure that contains the information of a virtual writer	2091
DDS_WaitSetProperty_t	<< <i>extension</i> >> (p. 1012) Specifies the DDS_WaitSet (p. 1371) behavior for multiple trigger events	2091
DDS_WcharSeq	Instantiates FooSeq (p. 2113) < DDS_Wchar (p. 1202) >	2093
DDS_WireProtocolQosPolicy	<< <i>extension</i> >> (p. 795) Specifies the wire-protocol-related attributes for the DDS_DomainParticipant (p. 77)	2093
DDS_WriteParams_t	<< <i>extension</i> >> (p. 1012) Input parameters for writing with FooDataWriter_write_w_params (p. 501), FooDataWriter_dispose_w_params (p. 504), FooDataWriter_register_instance_w_params (p. 493), FooDataWriter_unregister_instance_w_params (p. 496)	2101
DDS_WriterDataLifecycleQosPolicy	<< <i>spec</i> >> (p. 790) Controls how a DDS_DataWriter (p. 485) handles the lifecycle of the instances (keys) that it is registered to manage	2106
DDS_WstringSeq	Instantiates FooSeq (p. 2113) < DDS_Wchar (p. 1202)* >	2109
Foo	A representative user-defined data type	2109
FooBarReplier	Allows receiving requests and sending replies	2110
FooBarRequester	Allows sending requests and receiving replies	2111
FooBarSimpleReplier	A callback-based replier	2112
FooDataReader	<< <i>interface</i> >> (p. 1013) << <i>generic</i> >> (p. 1013) User data type-specific data reader	2112
FooDataWriter	<< <i>interface</i> >> (p. 1013) << <i>generic</i> >> (p. 1013) User data type specific data writer	2113

FooSeq	
<< <i>interface</i> >> (p. 1013) << <i>generic</i> >> (p. 1013) A type-safe, ordered collection of elements.	
The type of these elements is referred to in this documentation as Foo (p. 2109)	2113
FooTypeSupport	
<< <i>interface</i> >> (p. 1013) << <i>generic</i> >> (p. 1013) User data type specific interface	2114
NDDS_Config_LibraryVersion_t	
The version of a single library shipped as part of an RTI Connex distribution	2114
NDDS_Config_Logger	
<< <i>interface</i> >> (p. 1013) The singleton type used to configure RTI Connex logging	2116
NDDS_Config_LoggerDevice	
<< <i>interface</i> >> (p. 1013) Logging device interface. Use for user-defined logging devices	2116
NDDS_Config_LogMessage	
A log message's input parameter to a logging device, including the text and additional information	2117
NDDS_Config_Version_t	
<< <i>interface</i> >> (p. 1013) The version of an RTI Connex distribution	2119
NDDS_Transport_Address_t	
Addresses are stored individually as network-ordered bytes	2120
NDDS_Transport_Interface_t	
Storage for the description of a network interface used by a Transport Plugin	2120
NDDS_Transport_Property_t	
Base configuration structure that must be inherited by derived Transport Plugin classes	2122
NDDS_Transport_Shmem_Property_t	
Subclass of NDDS_Transport_Property_t (p. 2122) allowing specification of parameters that are specific to the shared-memory transport	2129
NDDS_Transport_Support	
<< <i>interface</i> >> (p. 1013) The utility class used to configure RTI Connex pluggable transports	2132
NDDS_Transport_UDP_WAN_CommPortsMappingInfo	
Type for storing UDP WAN communication ports	2132
NDDS_Transport_UDPv4_Property_t	
Configurable IPv4/UDP Transport-Plugin properties	2133
NDDS_Transport_UDPv4_WAN_Property_t	
Configurable IPv4/UDP WAN Transport-Plugin properties	2144
NDDS_Transport_UDPv6_Property_t	
Configurable IPv6/UDP Transport-Plugin properties	2154
NDDS_Transport_UUID	
Unequivocally identifies a transport plugin instance	2164
NDDS_Utility_HeapMonitoringParams_t	
Input parameters for enabling heap monitoring. They will be used for configuring the format of the snapshot	2164
NDDS_Utility_NetworkCaptureParams_t	
Input parameters for starting network capture	2165
RTI_Connext_Replier	
The type-independent version of a Replier	2167
RTI_Connext_ReplierListener	
Called when a FooBarReplier (p. 2110) has new available requests	2168
RTI_Connext_ReplierParams	
Contains the parameters for creating a FooBarReplier (p. 2110)	2169
RTI_Connext_Requester	
The type-independent version of a Requester	2173
RTI_Connext_RequesterParams	
Contains the parameters for creating a FooBarRequester (p. 2111)	2173
RTI_Connext_SimpleReplierListener	
The listener called by a SimpleReplier	2176

RTI_Connext_SimpleReplierParamsContains the parameters for creating a **FooBarSimpleReplier** (p. 2112) 2177**TransportAllocationSettings_t**

Allocation settings used by various internal buffers 2181

Chapter 4

File Index

4.1 File List

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

howto_prototypes.h	2183
transport_common.h	2184
transport_common_user.h	2189
transport_interface_user.h	2198
transport_shmem.h	2201
transport_udp.h	2205
transport_udp4.h	2217
transport_udp6.h	2229

Chapter 5

Module Documentation

5.1 Clock Selection

APIs related to clock selection.

APIs related to clock selection.

RTI Connex uses clocks to measure time and generate timestamps.

The middleware uses two clocks, an internal clock and an external clock. The internal clock is used to measure time and handles all timing in the middleware. The external clock is used solely to generate timestamps, such as the source timestamp and the reception timestamp, in addition to providing the time given by **DDS_DomainParticipant_get_↔current_time** (p. 138).

5.1.1 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 realtime 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".

5.1.2 Clock Selection Strategy

By default, both the internal and external clocks use the real-time clock. If you want your application to be robust to changes in the system time, you may use the monotonic clock as the internal clock, and leave the system clock as the external clock. Note, however, that this may slightly diminish performance in that both the send and receive paths may need to obtain times from both clocks. Since the monotonic clock is not available on all architectures, you may want to specify "monotonic,realtime" for the `internal_clock` (see the table below). By doing so, the middleware will attempt to use the monotonic clock if available, 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.

5.1.3 Configuring Clock Selection

To configure the clock selection, use the **PROPERTY** (p. 1306) QoS policy associated with the **DDS_DomainParticipant** (p. 77).

See also

DDS_PropertyQosPolicy (p. 1901)

The following table lists the supported clock selection properties.

Table 5.1 Clock Selection Properties

Property	Description
<code>dds.clock.external_clock</code>	Comma-delimited list of clocks to use for the external clock, in the order of preference. Valid clock names are "realtime", "system", and "monotonic". Default: "realtime"
<code>dds.clock.internal_clock</code>	Comma-delimited list of clocks to use for the internal clock, in the order of preference. Valid clock names are "realtime", "system", and "monotonic". Default: "realtime"

5.2 Domain Module

Contains the **DDS_DomainParticipant** (p. 77) class that acts as an entrypoint of RTI Connex and acts as a factory for many of the classes. The **DDS_DomainParticipant** (p. 77) also acts as a container for the other objects that make up RTI Connex.

Modules

- **DomainParticipantFactory**
DDS_DomainParticipantFactory (p. 32) entity and associated elements
- **DomainParticipants**
DDS_DomainParticipant (p. 77) entity and associated elements
- **Built-in Topics**
Built-in objects created by RTI Connex but accessible to the application.

5.2.1 Detailed Description

Contains the **DDS_DomainParticipant** (p. 77) class that acts as an entrypoint of RTI Connex and acts as a factory for many of the classes. The **DDS_DomainParticipant** (p. 77) also acts as a container for the other objects that make up RTI Connex.

5.3 DomainParticipantFactory

DDS_DomainParticipantFactory (p. 32) entity and associated elements

Modules

- **DomainParticipantConfigParams**
<<extension>> (p. 1012) *DDS_DomainParticipantConfigParams_t* (p. 1719)

Data Structures

- struct **DDS_DomainParticipantFactoryQos**
QoS policies supported by a DDS_DomainParticipantFactory (p. 32).

Macros

- **#define DDS_DomainParticipantFactoryQos_INITIALIZER**
Initializer for new QoS instances.
- **#define DDS_TheParticipantFactory DDS_DomainParticipantFactory_get_instance()**
Can be used as an alias for the singleton factory returned by the operation DDS_DomainParticipantFactory_get_instance (p. 38).

Typedefs

- typedef struct DDS_DomainParticipantFactoryImpl **DDS_DomainParticipantFactory**
<<singleton>> (p. 1013) *<<interface>>* (p. 1013) *Allows creation and destruction of DDS_DomainParticipant* (p. 77) *objects.*
- typedef **DDS_ReturnCode_t**(* **DDS_DomainParticipantFactory_RegisterTypeFunction**) (**DDS_DomainParticipant** *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. 1721) for equality.*
- **DDS_ReturnCode_t DDS_DomainParticipantFactoryQos_print** (const struct **DDS_DomainParticipantFactoryQos** *self)
*Prints this **DDS_DomainParticipantFactoryQos** (p. 1721) to stdout.*
- **DDS_ReturnCode_t DDS_DomainParticipantFactoryQos_to_string** (const struct **DDS_DomainParticipantFactoryQos** *self, char *string, **DDS_UnsignedLong** *string_size)
*Obtains a string representation of this **DDS_DomainParticipantQos** (p. 1726).*
- **DDS_ReturnCode_t DDS_DomainParticipantFactoryQos_to_string_w_params** (const struct **DDS_DomainParticipantFactoryQos** *self, char *string, **DDS_UnsignedLong** *string_size, const struct **DDS_DomainParticipantFactoryQos** *base, const struct **DDS_QosPrintFormat** *format)
*Obtains a string representation of this **DDS_DomainParticipantFactoryQos** (p. 1721).*
- **DDS_ReturnCode_t DDS_DomainParticipantFactoryQos_initialize** (struct **DDS_DomainParticipantFactoryQos** *self)
Initializer for new QoS instances.
- **DDS_ReturnCode_t DDS_DomainParticipantFactoryQos_finalize** (struct **DDS_DomainParticipantFactoryQos** *self)
*Free any dynamic memory allocated by the policies in this **DDS_DomainParticipantFactoryQos** (p. 1721).*
- **DDS_ReturnCode_t DDS_DomainParticipantFactoryQos_copy** (struct **DDS_DomainParticipantFactoryQos** *self, const struct **DDS_DomainParticipantFactoryQos** *source)
Copy the contents of the given QoS into this QoS.
- **DDS_DomainParticipantFactory * DDS_DomainParticipantFactory_get_instance** (void)
Gets the singleton instance of this class.
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_finalize_instance** (void)
<<extension>> (p. 1012) Destroys the singleton instance of this class.
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_set_default_participant_qos** (**DDS_DomainParticipantFactory** *self, const struct **DDS_DomainParticipantQos** *qos)
*Sets the default **DDS_DomainParticipantQos** (p. 1726) values for this domain participant factory.*
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_set_default_participant_qos_with_profile** (**DDS_DomainParticipantFactory** *self, const char *library_name, const char *profile_name)
*<<extension>> (p. 1012) Sets the default **DDS_DomainParticipantQos** (p. 1726) values for this domain participant factory based on the input XML QoS profile.*
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_get_default_participant_qos** (**DDS_DomainParticipantFactory** *self, struct **DDS_DomainParticipantQos** *qos)
*Initializes the **DDS_DomainParticipantQos** (p. 1726) instance with default values.*
- **DDS_DomainParticipant * DDS_DomainParticipantFactory_create_participant** (**DDS_DomainParticipantFactory** *self, **DDS_DomainId_t** domainId, const struct **DDS_DomainParticipantQos** *qos, const struct **DDS_DomainParticipantListener** *listener, **DDS_StatusMask** mask)
*Creates a new **DDS_DomainParticipant** (p. 77) object.*
- **DDS_DomainParticipant * DDS_DomainParticipantFactory_create_participant_with_profile** (**DDS_DomainParticipantFactory** *self, **DDS_DomainId_t** domainId, const char *library_name, const char *profile_name, const struct **DDS_DomainParticipantListener** *listener, **DDS_StatusMask** mask)
*<<extension>> (p. 1012) Creates a new **DDS_DomainParticipant** (p. 77) object using the **DDS_DomainParticipantQos** (p. 1726) associated with the input XML QoS profile.*
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_delete_participant** (**DDS_DomainParticipantFactory** *self, **DDS_DomainParticipant** *a_participant)
*Deletes an existing **DDS_DomainParticipant** (p. 77).*

- **DDS_DomainParticipant * DDS_DomainParticipantFactory_lookup_participant (DDS_DomainParticipantFactory *self, DDS_DomainId_t domainId)**
Locates an existing DDS_DomainParticipant (p. 77).
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_get_qos (DDS_DomainParticipantFactory *self, struct DDS_DomainParticipantFactoryQos *qos)**
Gets the value for participant factory QoS.
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_set_qos (DDS_DomainParticipantFactory *self, const struct DDS_DomainParticipantFactoryQos *qos)**
Sets the value for a participant factory QoS.
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_load_profiles (DDS_DomainParticipantFactory *self)**
<<extension>> (p. 1012) Loads the XML QoS profiles.
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_reload_profiles (DDS_DomainParticipantFactory *self)**
<<extension>> (p. 1012) Reloads the XML QoS profiles.
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_unload_profiles (DDS_DomainParticipantFactory *self)**
<<extension>> (p. 1012) Unloads the XML QoS profiles.
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_set_default_library (DDS_DomainParticipantFactory *self, const char *library_name)**
<<extension>> (p. 1012) Sets the default XML library for a DDS_DomainParticipantFactory (p. 32).
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_set_default_profile (DDS_DomainParticipantFactory *self, const char *library_name, const char *profile_name)**
<<extension>> (p. 1012) Sets the default XML profile for a DDS_DomainParticipantFactory (p. 32).
- **const char * DDS_DomainParticipantFactory_get_default_library (DDS_DomainParticipantFactory *self)**
<<extension>> (p. 1012) Gets the default XML library associated with a DDS_DomainParticipantFactory (p. 32).
- **const char * DDS_DomainParticipantFactory_get_default_profile (DDS_DomainParticipantFactory *self)**
<<extension>> (p. 1012) Gets the default XML profile associated with a DDS_DomainParticipantFactory (p. 32).
- **const char * DDS_DomainParticipantFactory_get_default_profile_library (DDS_DomainParticipantFactory *self)**
<<extension>> (p. 1012) Gets the library where the default XML profile is contained for a DDS_DomainParticipantFactory (p. 32).
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_get_participant_factory_qos_from_profile (DDS_DomainParticipantFactory *self, struct DDS_DomainParticipantFactoryQos *qos, const char *library_name, const char *profile_name)**
<<extension>> (p. 1012) Gets the DDS_DomainParticipantFactoryQos (p. 1721) values associated with the input XML QoS profile.
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_get_participant_qos_from_profile (DDS_DomainParticipantFactory *self, struct DDS_DomainParticipantQos *qos, const char *library_name, const char *profile_name)**
<<extension>> (p. 1012) Gets the DDS_DomainParticipantQos (p. 1726) values associated with the input XML QoS profile.
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_get_publisher_qos_from_profile (DDS_DomainParticipantFactory *self, struct DDS_PublisherQos *qos, const char *library_name, const char *profile_name)**
<<extension>> (p. 1012) Gets the DDS_PublisherQos (p. 1918) values associated with the input XML QoS profile.
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_get_subscriber_qos_from_profile (DDS_DomainParticipantFactory *self, struct DDS_SubscriberQos *qos, const char *library_name, const char *profile_name)**
<<extension>> (p. 1012) Gets the DDS_SubscriberQos (p. 2010) values associated with the input XML QoS profile.

- **DDS_ReturnCode_t DDS_DomainParticipantFactory_get_datareader_qos_from_profile** (**DDS_DomainParticipantFactory** *self, struct **DDS_DataReaderQos** *qos, const char *library_name, const char *profile_name)

<<extension>> (p. 1012) Gets the **DDS_DataReaderQos** (p. 1611) values associated with the input XML QoS profile.
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_get_datareader_qos_from_profile_w_topic_name** (**DDS_DomainParticipantFactory** *self, struct **DDS_DataReaderQos** *qos, const char *library_name, const char *profile_name, const char *topic_name)

<<extension>> (p. 1012) Gets the **DDS_DataReaderQos** (p. 1611) values associated with the input XML QoS profile while applying topic filters to the input topic name.
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_get_datawriter_qos_from_profile** (**DDS_DomainParticipantFactory** *self, struct **DDS_DataWriterQos** *qos, const char *library_name, const char *profile_name)

<<extension>> (p. 1012) Gets the **DDS_DataWriterQos** (p. 1666) values associated with the input XML QoS profile.
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_get_datawriter_qos_from_profile_w_topic_name** (**DDS_DomainParticipantFactory** *self, struct **DDS_DataWriterQos** *qos, const char *library_name, const char *profile_name, const char *topic_name)

<<extension>> (p. 1012) Gets the **DDS_DataWriterQos** (p. 1666) values associated with the input XML QoS profile while applying topic filters to the input topic name.
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_get_topic_qos_from_profile** (**DDS_DomainParticipantFactory** *self, struct **DDS_TopicQos** *qos, const char *library_name, const char *profile_name)

<<extension>> (p. 1012) Gets the **DDS_TopicQos** (p. 2042) values associated with the input XML QoS profile.
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_get_topic_qos_from_profile_w_topic_name** (**DDS_DomainParticipantFactory** *self, struct **DDS_TopicQos** *qos, const char *library_name, const char *profile_name, const char *topic_name)

<<extension>> (p. 1012) Gets the **DDS_TopicQos** (p. 2042) values associated with the input XML QoS profile while applying topic filters to the input topic name.
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_get_qos_profile_libraries** (**DDS_DomainParticipantFactory** *self, struct **DDS_StringSeq** *library_names)

<<extension>> (p. 1012) Gets the names of all XML QoS profile libraries associated with the **DDS_DomainParticipantFactory** (p. 32)
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_get_qos_profiles** (**DDS_DomainParticipantFactory** *self, struct **DDS_StringSeq** *profile_names, const char *library_name)

<<extension>> (p. 1012) Gets the names of all XML QoS profiles associated with the input XML QoS profile library.
- **DDS_ReturnCode_t DDS_DomainParticipantFactory_unregister_thread** (**DDS_DomainParticipantFactory** *self)

<<extension>> (p. 1012) Allows you to release thread-specific resources kept by RTI Connex on platforms that do not support automatic thread-specific storage (TSS) cleanup.
- **const DDS_TypeCode * DDS_DomainParticipantFactory_get_typecode_from_config** (**DDS_DomainParticipantFactory** *self, const char *type_name)

<<extension>> (p. 1012) Gets a **DDS_TypeCode** (p. 2071) from a definition provided in an XML configuration file.
- **DDS_DomainParticipant * DDS_DomainParticipantFactory_create_participant_from_config** (**DDS_DomainParticipantFactory** *self, const char *configuration_name)

<<extension>> (p. 1012) Creates a **DDS_DomainParticipant** (p. 77) given its configuration name from a description provided in an XML configuration file.
- **DDS_DomainParticipant * DDS_DomainParticipantFactory_create_participant_from_config_w_params** (**DDS_DomainParticipantFactory** *self, const char *configuration_name, const struct **DDS_DomainParticipantConfigParams_t** *params)

<<extension>> (p. 1012) Creates a **DDS_DomainParticipant** (p. 77) 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.
- **DDS_DomainParticipant * DDS_DomainParticipantFactory_lookup_participant_by_name** (**DDS_DomainParticipantFactory** *self, const char *participant_name)

<<extension>> (p. 1012) Looks up a **DDS_DomainParticipant** (p. 77) by its entity name in the **DDS_DomainParticipantFactory** (p. 32).

- **DDS_ReturnCode_t DDS_DomainParticipantFactory_register_type_support** (**DDS_DomainParticipantFactory** *self, **DDS_DomainParticipantFactory_RegisterTypeFunction** register_type_fcn, const char *type_name)

<<extension>> (p. 1012) Registers a **DDS_TypeSupport** (p. 214) with the **DDS_DomainParticipantFactory** (p. 32) to enable automatic registration if the corresponding type, should it be needed by a **DDS_DomainParticipant** (p. 77).

- **DDS_ReturnCode_t DDS_DomainParticipantFactory_get_participants** (**DDS_DomainParticipantFactory** *self, struct **DDS_DomainParticipantSeq** *participants)

<<extension>> (p. 1012) Allows the application to access all the participants the DomainParticipantFactory has.

- **DDS_ReturnCode_t DDS_DomainParticipantFactory_set_thread_factory** (**DDS_DomainParticipantFactory** *self, const struct **DDS_ThreadFactory** *thread_factory)

<<extension>> (p. 1012) Sets a **DDS_ThreadFactory** (p. 2027) in the **DDS_DomainParticipantFactory** (p. 32) that will be used to create the internal threads of the DDS middleware.

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 **DDS_DomainParticipant** (p. 77) from configuration using default parameters.

5.3.1 Detailed Description

DDS_DomainParticipantFactory (p. 32) entity and associated elements

5.3.2 Macro Definition Documentation

5.3.2.1 DDS_DomainParticipantFactoryQos_INITIALIZER

```
#define DDS_DomainParticipantFactoryQos_INITIALIZER
```

Initializer for new QoS instances.

New **DDS_DomainParticipantFactoryQos** (p. 1721) instances stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those contained QoS policies that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a new QoS structure is to initialize it on the stack at the time of its creation:

```
struct DDS_DomainParticipantFactoryQos myQos = DDS_DomainParticipantFactoryQos_INITIALIZER;
```

Note that the above assignment is not a substitute for calling **DDS_DomainParticipantFactory_get_qos** (p. 45); that function should be called subsequently to setting the QoS. **DDS_DomainParticipantFactoryQos_finalize** (p. 36) should be called to free the contained QoS policies that use dynamic memory:

```
struct DDS_DomainParticipantFactoryQos myQos = DDS_DomainParticipantFactoryQos_INITIALIZER;
```

```
DDS_DomainParticipantFactory_get_qos(myFactory, &myQos);
```

```
DDS_DomainParticipantFactory_set_qos(myFactory, &myQos);
```

```
DDS_DomainParticipantFactory_finalize(myFactory, &myQos);
```

See also

DDS_DomainParticipantFactory_get_qos (p. 45)

DDS_DomainParticipantFactoryQos_finalize (p. 36)

5.3.2.2 DDS_TheParticipantFactory

```
#define DDS_TheParticipantFactory DDS_DomainParticipantFactory_get_instance()
```

Can be used as an alias for the singleton factory returned by the operation **DDS_DomainParticipantFactory_get_instance** (p. 38).

See also

DDS_DomainParticipantFactory_get_instance (p. 38)

Examples

HelloWorld_publisher.c, and **HelloWorld_subscriber.c**.

5.3.3 Typedef Documentation

5.3.3.1 DDS_DomainParticipantFactory

```
typedef struct DDS_DomainParticipantFactoryImpl DDS_DomainParticipantFactory
```

<<***singleton***>> (p. 1013) <<***interface***>> (p. 1013) Allows creation and destruction of **DDS_DomainParticipant** (p. 77) objects.

The sole purpose of this class is to allow the creation and destruction of **DDS_DomainParticipant** (p. 77) objects. This class itself is a <<***singleton***>> (p. 1013), and accessed via the `get_instance()` function, and destroyed with `finalize_instance()` function.

A single application can participate in multiple domains by instantiating multiple **DDS_DomainParticipant** (p. 77) 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_WireProtocolQosPolicy::participant_id** (p. 2097)). 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 permissible 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

DDS_DomainParticipant (p. 77)

5.3.3.2 DDS_DomainParticipantFactory_RegisterTypeFunction

```
typedef DDS_ReturnCode_t (* DDS_DomainParticipantFactory_RegisterTypeFunction) ( DDS_DomainParticipant *participant, const char *type_name)
```

Prototype of a register type function.

Parameters

<i>participant</i>	<< <i>inout</i> >> (p. 1013) DDS_DomainParticipant (p. 77) participant the type is registered with.
<i>type_name</i>	<< <i>in</i> >> (p. 1013) Name the type is registered with.

Returns

One of the **Standard Return Codes** (p. 1221)

5.3.4 Function Documentation

5.3.4.1 DDS_DomainParticipantFactoryQos_equals()

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

Compares two **DDS_DomainParticipantFactoryQos** (p. 1721) for equality.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) This DomainParticipantFactoryQos.
<i>other</i>	<< <i>in</i> >> (p. 1013) The other DomainParticipantFactoryQos to be compared with this DomainParticipantFactoryQos.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two Qos objects are equal or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.3.4.2 DDS_DomainParticipantFactoryQos_print()

```
DDS_ReturnCode_t DDS_DomainParticipantFactoryQos_print (
    const struct DDS_DomainParticipantFactoryQos * self )
```

Prints this **DDS_DomainParticipantFactoryQos** (p. 1721) to stdout.

Only the differences between this **DDS_DomainParticipantFactoryQos** (p. 1721) and the documented default are printed. If you wish to print everything regardless, see **DDS_DomainParticipantFactoryQos_to_string_w_params** (p. 35). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1926).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_RETCODE_OK (p. 1222) if no error was encountered.

5.3.4.3 DDS_DomainParticipantFactoryQos_to_string()

```
DDS_ReturnCode_t DDS_DomainParticipantFactoryQos_to_string (
    const struct DDS_DomainParticipantFactoryQos * self,
    char * string,
    DDS_UnsignedLong * string_size )
```

Obtains a string representation of this **DDS_DomainParticipantQos** (p. 1726).

Only the differences between this **DDS_DomainParticipantFactoryQos** (p. 1721) and the documented default are printed to the string. If you wish to print everything regardless, see **DDS_DomainParticipantFactoryQos_to_string_w_params** (p. 35). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1926).

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. 1721) is written to the buffer.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>string</i>	<< <i>out</i> >> (p. 1013) The char buffer that will be used to store the string representation of this DDS_DomainParticipantFactoryQos (p. 1721). If NULL, this function will return the required length of this buffer through the <i>string_size</i> parameter.
<i>string_size</i>	<< <i>inout</i> >> (p. 1013) 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. 1222) if no error was encountered.

See also

DDS_DomainParticipantFactoryQos_to_string_w_params (p. 35)

5.3.4.4 DDS_DomainParticipantFactoryQos_to_string_w_params()

```
DDS_ReturnCode_t DDS_DomainParticipantFactoryQos_to_string_w_params (
    const struct DDS_DomainParticipantFactoryQos * self,
    char * string,
    DDS_UnsignedLong * string_size,
    const struct DDS_DomainParticipantFactoryQos * base,
    const struct DDS_QosPrintFormat * format )
```

Obtains a string representation of this **DDS_DomainParticipantFactoryQos** (p. 1721).

Only the differences between this **DDS_DomainParticipantFactoryQos** (p. 1721) and the **DDS_DomainParticipantFactoryQos** (p. 1721) supplied as the base are printed to the string.

It is possible to supply the sentinel value **DDS_DOMAINPARTICIPANTFACTORY_QOS_PRINT_ALL** (p. 160) as the base to print everything within the QoS.

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. 1721) is written to the buffer.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>string</i>	<< <i>out</i> >> (p. 1013) The char buffer that will be used to store the string representation of this DDS_DomainParticipantFactoryQos (p. 1721). If NULL, this function will return the required length of this buffer through the <i>string_size</i> parameter.
<i>string_size</i>	<< <i>inout</i> >> (p. 1013) 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.
<i>base</i>	<< <i>in</i> >> (p. 1013) The DDS_DomainParticipantFactoryQos (p. 1721) to be used as the base QoS profile. Only the differences with respect to this base profile will be included in the output string.
<i>format</i>	<< <i>in</i> >> (p. 1013) The DDS_QosPrintFormat (p. 1926) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 1222) if no error was encountered.

5.3.4.5 DDS_DomainParticipantFactoryQos_initialize()

```
DDS_ReturnCode_t DDS_DomainParticipantFactoryQos_initialize (
    struct DDS_DomainParticipantFactoryQos * self )
```

Initializer for new QoS instances.

New **DDS_DomainParticipantFactoryQos** (p. 1721) instances on heap should be initialized with this function before they are passed to any function. This step ensures that those contained QoS policies that use dynamic memory are properly initialized. This function does not allocate memory.

Calling this function is not a substitute for calling **DDS_DomainParticipantFactory_get_qos** (p. 45); that function should be called subsequently to setting the QoS of an existing factory. **DDS_DomainParticipantFactoryQos_finalize** (p. 36) should be called to free the contained QoS policies that use dynamic memory:

```
DDS_DomainParticipantFactoryQos *myQos = malloc(sizeof(struct DDS_DomainParticipantFactoryQos));

DDS_DomainParticipantQos_initialize(myQos);

DDS_DomainParticipantFactory_get_qos(myFactory, myQos);

DDS_DomainParticipantFactory_set_qos(myFactory, myQos);

DDS_DomainParticipantFactoryQos_finalize(myQos);

free(myQos);
```

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DomainParticipantFactory_get_qos (p. 45)

DDS_DomainParticipantFactoryQos_finalize (p. 36)

5.3.4.6 DDS_DomainParticipantFactoryQos_finalize()

```
DDS_ReturnCode_t DDS_DomainParticipantFactoryQos_finalize (
    struct DDS_DomainParticipantFactoryQos * self )
```

Free any dynamic memory allocated by the policies in this **DDS_DomainParticipantFactoryQos** (p. 1721).

Some QoS policies may use dynamic memory (regardless of whether the QoS itself is in dynamic memory). This function frees that memory but otherwise leaves this QoS unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

This function does not leave this object in an invalid state. It is permissible to clear a QoS and then subsequently allocate new dynamic memory in one or more of its QoS policies.

Note that if this QoS instance is stored in heap memory, calling this function will *not* call free() on it; the user is responsible for explicitly freeing any heap-based QoS instance after calling this function.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DomainParticipantFactoryQos_INITIALIZER (p. 31)

DDS_DomainParticipantFactoryQos_initialize (p. 36)

5.3.4.7 DDS_DomainParticipantFactoryQos_copy()

```
DDS_ReturnCode_t DDS_DomainParticipantFactoryQos_copy (
    struct DDS_DomainParticipantFactoryQos * self,
    const struct DDS_DomainParticipantFactoryQos * source )
```

Copy the contents of the given QoS into this QoS.

DDS_DomainParticipantFactoryQos (p. 1721) instances can use dynamic memory because of the sequences contained in some QoS policies. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). QoS to be copied from.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DomainParticipantFactoryQos_INITIALIZER (p. 31)

DDS_DomainParticipantFactoryQos_initialize (p. 36)

DDS_DomainParticipantFactoryQos_finalize (p. 36)

5.3.4.8 DDS_DomainParticipantFactory_get_instance()

```
DDS_DomainParticipantFactory * DDS_DomainParticipantFactory_get_instance (
    void )
```

Gets the singleton instance of this class.

DDS_TheParticipantFactory (p. 32) can be used as an alias for the singleton factory returned by this operation.

Returns

The singleton **DDS_DomainParticipantFactory** (p. 32) instance.

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this function while another thread may be simultaneously calling **DDS_DomainParticipantFactory_get_instance** (p. 38), **DDS_DomainParticipantFactory_finalize_instance** (p. 38), **DDS_TypeCodeFactory_get_instance** (p. 289), **DDS_GuardCondition_new** (p. 1376), **DDS_WaitSet_new** (p. 1380), **DDS_WaitSet_new_ex** (p. 1380), **DDS_GuardCondition_delete** (p. 1376), **DDS_WaitSet_delete** (p. 1381), **NDDS_Config_Logger_get_instance** (p. 1448), **NDDS_Config_Logger_get_output_device** (p. 1456), **NDDS_Config_Logger_set_output_device** (p. 1457), **DDS_TypeCodeFactory_finalize_instance** (p. 289), **NDDS_Utility_enable_network_capture**, or **NDDS_Utility_disable_network_capture**.

See also

DDS_TheParticipantFactory (p. 32)

5.3.4.9 DDS_DomainParticipantFactory_finalize_instance()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_finalize_instance (
    void )
```

<<**extension**>> (p. 1012) 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 function 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 **DDS_DomainParticipantFactory_↵
get_instance** (p. 38) will return a new lifecycle of the singleton.

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this function while another thread may be simultaneously calling **DDS_DomainParticipantFactory_get_instance** (p. 38), **DDS_DomainParticipantFactory_finalize_instance** (p. 38), **DDS_TypeCodeFactory_get_instance** (p. 289), **DDS_GuardCondition_new** (p. 1376), **DDS_WaitSet_↵
_new** (p. 1380), **DDS_WaitSet_new_ex** (p. 1380), **DDS_GuardCondition_delete** (p. 1376), **DDS_WaitSet_↵
delete** (p. 1381), **NDDS_Config_Logger_get_instance** (p. 1448), **NDDS_Config_Logger_get_output_device** (p. 1456), **NDDS_Config_Logger_set_output_device** (p. 1457), **DDS_TypeCodeFactory_finalize_instance** (p. 289), **NDDS_Utility_enable_network_capture**, or **NDDS_Utility_disable_network_capture**.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222)

See also

DDS_TheParticipantFactory (p. 32)

5.3.4.10 DDS_DomainParticipantFactory_set_default_participant_qos()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_set_default_participant_qos (
    DDS_DomainParticipantFactory * self,
    const struct DDS_DomainParticipantQos * qos )
```

Sets the default **DDS_DomainParticipantQos** (p. 1726) values for this domain participant factory.

This function may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>inout</i> >> (p. 1013) Qos to be filled up. The special value DDS_PARTICIPANT_QOS_DEFAULT (p. 65) may be passed as <i>qos</i> to indicate that the default QoS should be reset back to the initial values the factory would used if DDS_DomainParticipantFactory_set_default_participant_qos (p. 39) had never been called. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_PARTICIPANT_QOS_DEFAULT (p. 65)

DDS_DomainParticipantFactory_create_participant (p. 41)

5.3.4.11 DDS_DomainParticipantFactory_set_default_participant_qos_with_profile()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_set_default_participant_qos_with_profile (
    DDS_DomainParticipantFactory * self,
    const char * library_name,
    const char * profile_name )
```

<<*extension*>> (p. 1012) Sets the default **DDS_DomainParticipantQos** (p. 1726) values for this domain participant factory based on the input XML QoS profile.

This function may potentially allocate memory depending on the sequences contained in some QoS policies.

This default value will be used for newly created **DDS_DomainParticipant** (p. 77) if **DDS_PARTICIPANT_QOS_DEFAULT** (p. 65) is specified as the `qos` parameter when **DDS_DomainParticipantFactory_create_participant** (p. 41) is called.

Precondition

The **DDS_DomainParticipantQos** (p. 1726) 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. 1222)

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 **DDS_DomainParticipantFactory_set_default_participant_qos** (p. 39)

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< <i>in</i> >> (p. 1013) Library name containing the XML QoS profile. If <i>library_name</i> is null RTI Connex will use the default library (see DDS_DomainParticipantFactory_set_default_library (p. 48)).
<i>profile_name</i>	<< <i>in</i> >> (p. 1013) XML QoS Profile name. If <i>profile_name</i> is null RTI Connex will use the default profile (see DDS_DomainParticipantFactory_set_default_profile (p. 49)).

If the input profile cannot be found the function fails with **DDS_RETCODE_ERROR** (p. 1222).

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222)

See also

DDS_PARTICIPANT_QOS_DEFAULT (p. 65)

DDS_DomainParticipantFactory_create_participant_with_profile (p. 43)

5.3.4.12 DDS_DomainParticipantFactory_get_default_participant_qos()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_get_default_participant_qos (
    DDS_DomainParticipantFactory * self,
    struct DDS_DomainParticipantQos * qos )
```

Initializes the **DDS_DomainParticipantQos** (p. 1726) instance with default values.

The retrieved `qos` will match the set of values specified on the last successful call to **DDS_DomainParticipantFactory_set_default_participant_qos** (p. 39), or **DDS_DomainParticipantFactory_set_default_participant_qos_with_profile** (p. 40), or else, if the call was never made, the default values listed in **DDS_DomainParticipantQos** (p. 1726).

This function may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>out</i> >> (p. 1013) the domain participant's QoS Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_PARTICIPANT_QOS_DEFAULT (p. 65)

DDS_DomainParticipantFactory_create_participant (p. 41)

5.3.4.13 DDS_DomainParticipantFactory_create_participant()

```
DDS_DomainParticipant * DDS_DomainParticipantFactory_create_participant (
    DDS_DomainParticipantFactory * self,
```

```

    DDS_DomainId_t domainId,
    const struct DDS_DomainParticipantQos * qos,
    const struct DDS_DomainParticipantListener * listener,
    DDS_StatusMask mask )

```

Creates a new **DDS_DomainParticipant** (p. 77) object.

Precondition

The specified QoS policies must be consistent or the operation will fail and no **DDS_DomainParticipant** (p. 77) 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. 2093)). 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).

Precondition

If `listener` is specified, none of the listener callback functions can be NULL.

MT Safety:

Safe.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>domainId</i>	<< <i>in</i> >> (p. 1013) ID of the domain that the application intends to join. [range] [≥ 0], and does not violate guidelines stated in DDS_RtpsWellKnownPorts_t (p. 1978).
<i>qos</i>	<< <i>in</i> >> (p. 1013) the DomainParticipant's QoS. The special value DDS_PARTICIPANT_QOS_DEFAULT (p. 65) can be used to indicate that the DDS_DomainParticipant (p. 77) should be created with the default DDS_DomainParticipantQos (p. 1726) set in the DDS_DomainParticipantFactory (p. 32). Cannot be NULL.
<i>listener</i>	<< <i>in</i> >> (p. 1013) the domain participant's listener.
<i>mask</i>	<< <i>in</i> >> (p. 1013). Changes of communication status to be invoked on the listener. See DDS_StatusMask (p. 1226).

Returns

domain participant or NULL on failure

See also

Specifying QoS on entities (p. 1244) for information on setting QoS before entity creation

DDS_DomainParticipantQos (p. 1726) for rules on consistency among QoS

DDS_PARTICIPANT_QOS_DEFAULT (p. 65)

NDDS_DISCOVERY_PEERS (p. 1352)

DDS_DomainParticipantFactory_create_participant_with_profile (p. 43)

DDS_DomainParticipantFactory_get_default_participant_qos (p. 41)

DDS_DomainParticipant_set_listener (p. 154)

Examples

HelloWorld_publisher.c, and **HelloWorld_subscriber.c**.

5.3.4.14 DDS_DomainParticipantFactory_create_participant_with_profile()

```
DDS_DomainParticipant * DDS_DomainParticipantFactory_create_participant_with_profile (
    DDS_DomainParticipantFactory * self,
    DDS_DomainId_t domainId,
    const char * library_name,
    const char * profile_name,
    const struct DDS_DomainParticipantListener * listener,
    DDS_StatusMask mask )
```

<<**extension**>> (p. 1012) Creates a new **DDS_DomainParticipant** (p. 77) object using the **DDS_DomainParticipantQos** (p. 1726) associated with the input XML QoS profile.

Precondition

The **DDS_DomainParticipantQos** (p. 1726) in the input profile must be consistent, or the operation will fail and no **DDS_DomainParticipant** (p. 77) 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. 2093)). 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).

Precondition

if *listener* is specified, none of the listener callback functions can be NULL.

MT Safety:

Safe.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>domainId</i>	<< <i>in</i> >> (p. 1013) ID of the domain that the application intends to join. [range] [≥ 0], and does not violate guidelines stated in DDS_RtpsWellKnownPorts_t (p. 1978).
<i>library_name</i>	<< <i>in</i> >> (p. 1013) Library name containing the XML QoS profile. If <i>library_name</i> is null RTI Connex will use the default library (see DDS_DomainParticipantFactory_set_default_library (p. 48)).
<i>profile_name</i>	<< <i>in</i> >> (p. 1013) XML QoS Profile name. If <i>profile_name</i> is null RTI Connex will use the default profile (see DDS_DomainParticipantFactory_set_default_profile (p. 49)).
<i>listener</i>	<< <i>in</i> >> (p. 1013) the DomainParticipant's listener.
<i>mask</i>	<< <i>in</i> >> (p. 1013). Changes of communication status to be invoked on the listener. See DDS_StatusMask (p. 1226).

Returns

domain participant or NULL on failure

See also

Specifying QoS on entities (p. 1244) for information on setting QoS before entity creation
DDS_DomainParticipantQos (p. 1726) for rules on consistency among QoS
DDS_PARTICIPANT_QOS_DEFAULT (p. 65)
NDDS_DISCOVERY_PEERS (p. 1352)
DDS_DomainParticipantFactory_create_participant() (p. 41)
DDS_DomainParticipantFactory_get_default_participant_qos() (p. 41)
DDS_DomainParticipant_set_listener() (p. 154)

5.3.4.15 DDS_DomainParticipantFactory_delete_participant()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_delete_participant (
    DDS_DomainParticipantFactory * self,
    DDS_DomainParticipant * a_participant )
```

Deletes an existing **DDS_DomainParticipant** (p. 77).

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. 1222).

Postcondition

Listener installed on the **DDS_DomainParticipant** (p. 77) will not be called after this function 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.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>a_participant</i>	<< <i>in</i> >> (p. 1013) DDS_DomainParticipant (p. 77) to be deleted.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

Examples

HelloWorld_publisher.c, and **HelloWorld_subscriber.c**.

5.3.4.16 DDS_DomainParticipantFactory_lookup_participant()

```
DDS_DomainParticipant * DDS_DomainParticipantFactory_lookup_participant (
    DDS_DomainParticipantFactory * self,
    DDS_DomainId_t domainId )
```

Locates an existing **DDS_DomainParticipant** (p. 77).

If no such **DDS_DomainParticipant** (p. 77) exists, the operation will return NULL value.

If multiple **DDS_DomainParticipant** (p. 77) entities belonging to that domainId exist, then the operation will return one of them. It is not specified which one.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>domainId</i>	<< <i>in</i> >> (p. 1013) ID of the domain participant to lookup.

Returns

domain participant if it exists, or NULL

5.3.4.17 DDS_DomainParticipantFactory_get_qos()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_get_qos (
    DDS_DomainParticipantFactory * self,
    struct DDS_DomainParticipantFactoryQos * qos )
```

Gets the value for participant factory QoS.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>inout</i> >> (p. 1013) QoS to be filled up. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.3.4.18 DDS_DomainParticipantFactory_set_qos()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_set_qos (
    DDS_DomainParticipantFactory * self,
    const struct DDS_DomainParticipantFactoryQos * qos )
```

Sets the value for a participant factory QoS.

The **DDS_DomainParticipantFactoryQos::entity_factory** (p.1722) can be changed. The other policies are immutable.

Note that despite having QoS, the **DDS_DomainParticipantFactory** (p. 32) is not an **DDS_Entity** (p. 1360).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>in</i> >> (p. 1013) Set of policies to be applied to DDS_DomainParticipantFactory (p. 32). Policies must be consistent. Immutable policies can only be changed before calling any other RTI Connex functions except for DDS_DomainParticipantFactory_get_qos (p. 45) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 1222) if immutable policy is changed, or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222) if policies are inconsistent

See also

DDS_DomainParticipantFactoryQos (p. 1721) for rules on consistency among QoS

5.3.4.19 DDS_DomainParticipantFactory_load_profiles()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_load_profiles (
    DDS_DomainParticipantFactory * self )
```

<<*extension*>> (p. 1012) Loads the XML QoS profiles.

The XML QoS profiles are loaded implicitly after the first **DDS_DomainParticipant** (p. 77) is created or explicitly, after a call to this function.

This has the same effect as **DDS_DomainParticipantFactory_reload_profiles()** (p. 47).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_ProfileQosPolicy (p. 1898)

5.3.4.20 DDS_DomainParticipantFactory_reload_profiles()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_reload_profiles (
    DDS_DomainParticipantFactory * self )
```

<<*extension*>> (p. 1012) Reloads the XML QoS profiles.

The XML QoS profiles are loaded implicitly after the first **DDS_DomainParticipant** (p. 77) is created or explicitly, after a call to this function.

This has the same effect as **DDS_DomainParticipantFactory_load_profiles()** (p. 46).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_ProfileQosPolicy (p. 1898)

5.3.4.21 DDS_DomainParticipantFactory_unload_profiles()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_unload_profiles (
    DDS_DomainParticipantFactory * self )
```

<<**extension**>> (p. 1012) Unloads the XML QoS profiles.

The resources associated with the XML QoS profiles are freed. Any reference to the profiles after calling this function will fail with an error.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_ProfileQosPolicy (p. 1898)

5.3.4.22 DDS_DomainParticipantFactory_set_default_library()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_set_default_library (
    DDS_DomainParticipantFactory * self,
    const char * library_name )
```

<<**extension**>> (p. 1012) Sets the default XML library for a **DDS_DomainParticipantFactory** (p. 32).

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. 1722)) 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

DDS_DomainParticipantFactory_set_default_profile (p. 49) for more information.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) Library name. If <code>library_name</code> is NULL any previous default is unset.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DomainParticipantFactory_get_default_library (p. 49)

5.3.4.23 DDS_DomainParticipantFactory_set_default_profile()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_set_default_profile (
    DDS_DomainParticipantFactory * self,
    const char * library_name,
    const char * profile_name )
```

<<**extension**>> (p. 1012) Sets the default XML profile for a **DDS_DomainParticipantFactory** (p. 32).

This function 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 function. When calling a **DDS_DomainParticipantFactory** (p. 32) function 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 function does not set the default QoS for newly created DomainParticipants; for this functionality, use **DDS_DomainParticipantFactory_set_default_participant_qos_with_profile** (p. 40) (you may pass in NULL after having called `set_default_profile()`).

Note: if the profile set with this function no longer exists after reloading the QoS profiles (for example, by changing **DDS_DomainParticipantFactoryQos::profile** (p. 1722)) 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

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) The library name containing the profile.
<i>profile_name</i>	<< in >> (p. 1013) The profile name. If <code>profile_name</code> is NULL any previous default is unset.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DomainParticipantFactory_get_default_profile (p. 50)

DDS_DomainParticipantFactory_get_default_profile_library (p. 51)

5.3.4.24 DDS_DomainParticipantFactory_get_default_library()

```
const char * DDS_DomainParticipantFactory_get_default_library (
    DDS_DomainParticipantFactory * self )
```

<<**extension**>> (p. 1012) Gets the default XML library associated with a **DDS_DomainParticipantFactory** (p. 32).

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.

Returns

The returned library name is determined as follows:

- If it was previously set with **DDS_DomainParticipantFactory_set_default_library** (p. 48), 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

DDS_DomainParticipantFactory_set_default_library (p. 48)

5.3.4.25 DDS_DomainParticipantFactory_get_default_profile()

```
const char * DDS_DomainParticipantFactory_get_default_profile (
    DDS_DomainParticipantFactory * self )
```

<<**extension**>> (p. 1012) Gets the default XML profile associated with a **DDS_DomainParticipantFactory** (p. 32).

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.

Returns

The returned profile name is determined as follows:

- If it was previously set with **DDS_DomainParticipantFactory_set_default_profile** (p. 49), 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

DDS_DomainParticipantFactory_set_default_profile (p. 49)

5.3.4.26 DDS_DomainParticipantFactory_get_default_profile_library()

```
const char * DDS_DomainParticipantFactory_get_default_profile_library (
    DDS_DomainParticipantFactory * self )
```

<<*extension*>> (p. 1012) Gets the library where the default XML profile is contained for a **DDS_DomainParticipantFactory** (p. 32).

The default profile library is automatically set when **DDS_DomainParticipantFactory_set_default_profile** (p. 49) is called.

This library can be different than the **DDS_DomainParticipantFactory** (p. 32) default library (see **DDS_DomainParticipantFactory_get_default_library** (p. 49)).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

The default profile library for the profile returned by **DDS_DomainParticipantFactory_get_default_profile** (p. 50), or NULL if that profile is NULL.

See also

DDS_DomainParticipantFactory_get_default_profile (p. 50)

5.3.4.27 DDS_DomainParticipantFactory_get_participant_factory_qos_from_profile()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_get_participant_factory_qos_from_profile (
    DDS_DomainParticipantFactory * self,
    struct DDS_DomainParticipantFactoryQos * qos,
    const char * library_name,
    const char * profile_name )
```

<<*extension*>> (p. 1012) Gets the **DDS_DomainParticipantFactoryQos** (p. 1721) values associated with the input XML QoS profile.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>out</i> >> (p. 1013) Qos to be filled up. Cannot be NULL.
<i>library_name</i>	<< <i>in</i> >> (p. 1013) Library name containing the XML QoS profile. If library_name is null RTI Connex will use the default library (see DDS_DomainParticipantFactory_set_default_library (p. 48)).
<i>profile_name</i>	<< <i>in</i> >> (p. 1013) XML QoS Profile name. If profile_name is null RTI Connex will use the default profile (see DDS_DomainParticipantFactory_set_default_profile (p. 49)).

If the input profile cannot be found, the function fails with **DDS_RETCODE_ERROR** (p. 1222).

Returns

One of the **Standard Return Codes** (p. 1221)

5.3.4.28 DDS_DomainParticipantFactory_get_participant_qos_from_profile()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_get_participant_qos_from_profile (
    DDS_DomainParticipantFactory * self,
    struct DDS_DomainParticipantQos * qos,
    const char * library_name,
    const char * profile_name )
```

<<*extension*>> (p. 1012) Gets the **DDS_DomainParticipantQos** (p. 1726) values associated with the input XML QoS profile.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>out</i> >> (p. 1013) Qos to be filled up. Cannot be NULL.
<i>library_name</i>	<< <i>in</i> >> (p. 1013) Library name containing the XML QoS profile. If library_name is null RTI Connex will use the default library (see DDS_DomainParticipantFactory_set_default_library (p. 48)).
<i>profile_name</i>	<< <i>in</i> >> (p. 1013) XML QoS Profile name. If profile_name is null RTI Connex will use the default profile (see DDS_DomainParticipantFactory_set_default_profile (p. 49)).

If the input profile cannot be found, the function fails with **DDS_RETCODE_ERROR** (p. 1222).

Returns

One of the **Standard Return Codes** (p. 1221)

5.3.4.29 DDS_DomainParticipantFactory_get_publisher_qos_from_profile()

```

DDS_ReturnCode_t DDS_DomainParticipantFactory_get_publisher_qos_from_profile (
    DDS_DomainParticipantFactory * self,
    struct DDS_PublisherQos * qos,
    const char * library_name,
    const char * profile_name )

```

<<**extension**>> (p. 1012) Gets the **DDS_PublisherQos** (p. 1918) values associated with the input XML QoS profile.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< out >> (p. 1013) Qos to be filled up. Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) Library name containing the XML QoS profile. If library_name is null RTI Connexnt will use the default library (see DDS_DomainParticipantFactory_set_default_library (p. 48)).
<i>profile_name</i>	<< in >> (p. 1013) XML QoS Profile name. If profile_name is null RTI Connexnt will use the default profile (see DDS_DomainParticipantFactory_set_default_profile (p. 49)).

If the input profile cannot be found, the function fails with **DDS_RETCODE_ERROR** (p. 1222).

Returns

One of the **Standard Return Codes** (p. 1221)

5.3.4.30 DDS_DomainParticipantFactory_get_subscriber_qos_from_profile()

```

DDS_ReturnCode_t DDS_DomainParticipantFactory_get_subscriber_qos_from_profile (
    DDS_DomainParticipantFactory * self,
    struct DDS_SubscriberQos * qos,
    const char * library_name,
    const char * profile_name )

```

<<**extension**>> (p. 1012) Gets the **DDS_SubscriberQos** (p. 2010) values associated with the input XML QoS profile.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< out >> (p. 1013) Qos to be filled up. Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) Library name containing the XML QoS profile. If library_name is null RTI Connexnt will use the default library (see DDS_DomainParticipantFactory_set_default_library (p. 48)).
<i>profile_name</i>	<< in >> (p. 1013) XML QoS Profile name. If profile_name is null RTI Connexnt will use the default profile (see DDS_DomainParticipantFactory_set_default_profile (p. 49)).

If the input profile cannot be found, the function fails with **DDS_RETCODE_ERROR** (p. 1222).

Returns

One of the **Standard Return Codes** (p. 1221)

5.3.4.31 DDS_DomainParticipantFactory_get_datareader_qos_from_profile()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_get_datareader_qos_from_profile (
    DDS_DomainParticipantFactory * self,
    struct DDS_DataReaderQos * qos,
    const char * library_name,
    const char * profile_name )
```

<<**extension**>> (p. 1012) Gets the **DDS_DataReaderQos** (p. 1611) values associated with the input XML QoS profile.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< out >> (p. 1013) Qos to be filled up. Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) Library name containing the XML QoS profile. If library_name is null RTI Connexx will use the default library (see DDS_DomainParticipantFactory_set_default_library (p. 48)).
<i>profile_name</i>	<< in >> (p. 1013) XML QoS Profile name. If profile_name is null RTI Connexx will use the default profile (see DDS_DomainParticipantFactory_set_default_profile (p. 49)).

If the input profile cannot be found, the function fails with **DDS_RETCODE_ERROR** (p. 1222).

Returns

One of the **Standard Return Codes** (p. 1221)

5.3.4.32 DDS_DomainParticipantFactory_get_datareader_qos_from_profile_w_topic_name()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_get_datareader_qos_from_profile_w_topic_name (
    DDS_DomainParticipantFactory * self,
    struct DDS_DataReaderQos * qos,
    const char * library_name,
    const char * profile_name,
    const char * topic_name )
```

<<**extension**>> (p. 1012) Gets the **DDS_DataReaderQos** (p. 1611) values associated with the input XML QoS profile while applying topic filters to the input topic name.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>out</i> >> (p. 1013) Qos to be filled up. Cannot be NULL.
<i>library_name</i>	<< <i>in</i> >> (p. 1013) Library name containing the XML QoS profile. If library_name is null RTI Connex will use the default library (see DDS_DomainParticipantFactory_set_default_library (p. 48)).
<i>profile_name</i>	<< <i>in</i> >> (p. 1013) XML QoS Profile name. If profile_name is null RTI Connex will use the default profile (see DDS_DomainParticipantFactory_set_default_profile (p. 49)).
<i>topic_name</i>	<< <i>in</i> >> (p. 1013) Topic name that will be evaluated against the topic_filter attribute in the XML QoS profile. If topic_name is null, RTI Connex will match only QoSs without explicit topic_filter expressions.

If the input profile cannot be found, the function fails with **DDS_RETCODE_ERROR** (p. 1222).

Returns

One of the **Standard Return Codes** (p. 1221)

5.3.4.33 DDS_DomainParticipantFactory_get_datawriter_qos_from_profile()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_get_datawriter_qos_from_profile (
    DDS_DomainParticipantFactory * self,
    struct DDS_DataWriterQos * qos,
    const char * library_name,
    const char * profile_name )
```

<<*extension*>> (p. 1012) Gets the **DDS_DataWriterQos** (p. 1666) values associated with the input XML QoS profile.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>out</i> >> (p. 1013) Qos to be filled up. Cannot be NULL.
<i>library_name</i>	<< <i>in</i> >> (p. 1013) Library name containing the XML QoS profile. If library_name is null RTI Connex will use the default library (see DDS_DomainParticipantFactory_set_default_library (p. 48)).
<i>profile_name</i>	<< <i>in</i> >> (p. 1013) XML QoS Profile name. If profile_name is null RTI Connex will use the default profile (see DDS_DomainParticipantFactory_set_default_profile (p. 49)).

If the input profile cannot be found, the function fails with **DDS_RETCODE_ERROR** (p. 1222).

Returns

One of the **Standard Return Codes** (p. 1221)

5.3.4.34 DDS_DomainParticipantFactory_get_datawriter_qos_from_profile_w_topic_name()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_get_datawriter_qos_from_profile_w_topic_name (
    DDS_DomainParticipantFactory * self,
    struct DDS_DataWriterQos * qos,
    const char * library_name,
    const char * profile_name,
    const char * topic_name )
```

<<**extension**>> (p. 1012) Gets the **DDS_DataWriterQos** (p. 1666) values associated with the input XML QoS profile while applying topic filters to the input topic name.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< out >> (p. 1013) Qos to be filled up. Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) Library name containing the XML QoS profile. If library_name is null RTI Connexnt will use the default library (see DDS_DomainParticipantFactory_set_default_library (p. 48)).
<i>profile_name</i>	<< in >> (p. 1013) XML QoS Profile name. If profile_name is null RTI Connexnt will use the default profile (see DDS_DomainParticipantFactory_set_default_profile (p. 49)).
<i>topic_name</i>	<< in >> (p. 1013) Topic name that will be evaluated against the topic_filter attribute in the XML QoS profile. If topic_name is null, RTI Connexnt will match only QoSs without explicit topic_filter expressions.

If the input profile cannot be found, the function fails with **DDS_RETCODE_ERROR** (p. 1222).

Returns

One of the **Standard Return Codes** (p. 1221)

5.3.4.35 DDS_DomainParticipantFactory_get_topic_qos_from_profile()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_get_topic_qos_from_profile (
    DDS_DomainParticipantFactory * self,
    struct DDS_TopicQos * qos,
    const char * library_name,
    const char * profile_name )
```

<<**extension**>> (p. 1012) Gets the **DDS_TopicQos** (p. 2042) values associated with the input XML QoS profile.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< out >> (p. 1013) Qos to be filled up. Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) Library name containing the XML QoS profile. If library_name is null RTI Connexnt will use the default library (see DDS_DomainParticipantFactory_set_default_library (p. 48)).
<i>profile_name</i>	<< in >> (p. 1013) XML QoS Profile name. If profile_name is null RTI Connexnt will use the default profile (see DDS_DomainParticipantFactory_set_default_profile (p. 49)).

If the input profile cannot be found, the function fails with **DDS_RETCODE_ERROR** (p. 1222).

Returns

One of the **Standard Return Codes** (p. 1221)

5.3.4.36 DDS_DomainParticipantFactory_get_topic_qos_from_profile_w_topic_name()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_get_topic_qos_from_profile_w_topic_name (
    DDS_DomainParticipantFactory * self,
    struct DDS_TopicQos * qos,
    const char * library_name,
    const char * profile_name,
    const char * topic_name )
```

<<**extension**>> (p. 1012) Gets the **DDS_TopicQos** (p. 2042) values associated with the input XML QoS profile while applying topic filters to the input topic name.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< out >> (p. 1013) Qos to be filled up. Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) Library name containing the XML QoS profile. If library_name is null RTI Connexnt will use the default library (see DDS_DomainParticipantFactory_set_default_library (p. 48)).
<i>profile_name</i>	<< in >> (p. 1013) XML QoS Profile name. If profile_name is null RTI Connexnt will use the default profile (see DDS_DomainParticipantFactory_set_default_profile (p. 49)).
<i>topic_name</i>	<< in >> (p. 1013) Topic name that will be evaluated against the topic_filter attribute in the XML QoS profile. If topic_name is null, RTI Connexnt will match only QoSs without explicit topic_filter expressions.

If the input profile cannot be found, the function fails with **DDS_RETCODE_ERROR** (p. 1222).

Returns

One of the **Standard Return Codes** (p. 1221)

5.3.4.37 DDS_DomainParticipantFactory_get_qos_profile_libraries()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_get_qos_profile_libraries (
    DDS_DomainParticipantFactory * self,
    struct DDS_StringSeq * library_names )
```

<<**extension**>> (p. 1012) Gets the names of all XML QoS profile libraries associated with the **DDS_DomainParticipantFactory** (p. 32)

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>library_names</i>	<< <i>out</i> >> (p. 1013) DDS_StringSeq (p. 2005) to be filled with names of XML QoS profile libraries. Cannot be NULL.

5.3.4.38 DDS_DomainParticipantFactory_get_qos_profiles()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_get_qos_profiles (
    DDS_DomainParticipantFactory * self,
    struct DDS_StringSeq * profile_names,
    const char * library_name )
```

<<*extension*>> (p. 1012) Gets the names of all XML QoS profiles associated with the input XML QoS profile library.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>profile_names</i>	<< <i>out</i> >> (p. 1013) DDS_StringSeq (p. 2005) to be filled with names of XML QoS profiles. Cannot be NULL.
<i>library_name</i>	<< <i>in</i> >> (p. 1013) Library name containing the XML QoS profile. If <i>library_name</i> is null RTI Connexx will use the default library (see DDS_DomainParticipantFactory_set_default_library (p. 48)).

5.3.4.39 DDS_DomainParticipantFactory_unregister_thread()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_unregister_thread (
    DDS_DomainParticipantFactory * self )
```

<<*extension*>> (p. 1012) Allows you to release thread-specific resources kept by RTI Connexx 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 Connexx 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.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.3.4.40 DDS_DomainParticipantFactory_get_typecode_from_config()

```
const DDS_TypeCode * DDS_DomainParticipantFactory_get_typecode_from_config (
    DDS_DomainParticipantFactory * self,
    const char * type_name )
```

<<**extension**>> (p. 1012) Gets a **DDS_TypeCode** (p. 2071) from a definition provided in an XML configuration file.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>type_name</i>	<< <i>in</i> >> (p. 1013) Name of the type in the configuration file.

Returns

The **DDS_TypeCode** (p. 2071) or NULL if a type with that name doesn't exist or an error occurs.

5.3.4.41 DDS_DomainParticipantFactory_create_participant_from_config()

```
DDS_DomainParticipant * DDS_DomainParticipantFactory_create_participant_from_config (
    DDS_DomainParticipantFactory * self,
    const char * configuration_name )
```

<<**extension**>> (p. 1012) Creates a **DDS_DomainParticipant** (p. 77) given its configuration name from a description provided in an XML configuration file.

This operation creates a **DDS_DomainParticipant** (p. 77) registering all the necessary data types and creating all the contained entities (**DDS_Topic** (p. 175), **DDS_Publisher** (p. 443), **DDS_Subscriber** (p. 572), **DDS_DataWriter** (p. 485), **DDS_DataReader** (p. 615)) 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 **DDS_DomainParticipant** (p. 77) can be retrieved with the help of lookup operations such as: **DDS_DomainParticipant_lookup_datareader_by_name** (p. 159).

This operation is equivalent to call **DDS_DomainParticipantFactory_create_participant_from_config_w_params** (p. 60) passing **DDS_PARTICIPANT_CONFIG_PARAMS_DEFAULT** (p. 65) as parameters.

MT Safety:

Safe.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>configuration_name</i>	<< <i>in</i> >> (p. 1013) Name of the participant configuration in the XML file.

Returns

The created **DDS_DomainParticipant** (p. 77) or NULL on error.

See also

DDS_DomainParticipantFactory_lookup_participant_by_name (p. 61)

DDS_DomainParticipant_lookup_topicdescription (p. 127)

DDS_DomainParticipant_lookup_publisher_by_name (p. 156)

DDS_DomainParticipant_lookup_subscriber_by_name (p. 157)

DDS_DomainParticipant_lookup_datareader_by_name (p. 159)

DDS_DomainParticipant_lookup_datawriter_by_name (p. 158)

DDS_Publisher_lookup_datawriter_by_name (p. 469)

DDS_Subscriber_lookup_datareader_by_name (p. 598)

5.3.4.42 DDS_DomainParticipantFactory_create_participant_from_config_w_params()

```
DDS_DomainParticipant * DDS_DomainParticipantFactory_create_participant_from_config_w_params (
    DDS_DomainParticipantFactory * self,
    const char * configuration_name,
    const struct DDS_DomainParticipantConfigParams_t * params )
```

<<extension>> (p. 1012) Creates a **DDS_DomainParticipant** (p. 77) 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 **DDS_DomainParticipant** (p. 77) the same way specified in **DDS_DomainParticipantFactory_create_participant_from_config** (p. 59).

In addition, this operation allows overriding the domain ID, participant name, and entities QoS specified in the configuration.

MT Safety:

Safe.

Parameters

<i>self</i>	<<in>> (p. 1013) Cannot be NULL.
<i>configuration_name</i>	<<in>> (p. 1013) Name of the participant configuration in the XML file.
<i>params</i>	<<in>> (p. 1013) input parameters that allow changing some properties of the configuration referred to by <i>configuration_name</i> .

Returns

The created **DDS_DomainParticipant** (p. 77) or NULL on error.

5.3.4.43 DDS_DomainParticipantFactory_lookup_participant_by_name()

```
DDS_DomainParticipant * DDS_DomainParticipantFactory_lookup_participant_by_name (
    DDS_DomainParticipantFactory * self,
    const char * participant_name )
```

<<extension>> (p. 1012) Looks up a **DDS_DomainParticipant** (p. 77) by its entity name in the **DDS_DomainParticipantFactory** (p. 32).

Every **DDS_DomainParticipant** (p. 77) in the system has an entity name which is configured and stored in the Entity Name policy, **ENTITY_NAME** (p. 1290).

This operation retrieves a **DDS_DomainParticipant** (p. 77) within the **DDS_DomainParticipantFactory** (p. 32) given the entity's name. If there are several **DDS_DomainParticipant** (p. 77) with the same name within the **DDS_DomainParticipantFactory** (p. 32) this function returns the first matching occurrence.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>participant_name</i>	<< <i>in</i> >> (p. 1013) Entity name of the DDS_DomainParticipant (p. 77). Cannot be NULL.

Returns

The first **DDS_DomainParticipant** (p. 77) found with the specified name or NULL if it is not found.

5.3.4.44 DDS_DomainParticipantFactory_register_type_support()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_register_type_support (
    DDS_DomainParticipantFactory * self,
    DDS_DomainParticipantFactory_RegisterTypeFunction register_type_fcn,
    const char * type_name )
```

<<*extension*>> (p. 1012) Registers a **DDS_TypeSupport** (p. 214) with the **DDS_DomainParticipantFactory** (p. 32) to enable automatic registration if the corresponding type, should it be needed by a **DDS_DomainParticipant** (p. 77).

Types referred by the **DDS_Topic** (p. 175) entities within a **DDS_DomainParticipant** (p. 77) must be registered with the **DDS_DomainParticipant** (p. 77).

Type registration in a **DDS_DomainParticipant** (p. 77) is performed by a call to the **FooTypeSupport_register_type** (p. 221) operation. This can be done directly from the application code or indirectly by the RTI Connex infrastructure as a result of parsing an XML configuration file that refers to that type.

The **DDS_DomainParticipantFactory_register_type_support** (p. 62) operation provides the **DDS_DomainParticipantFactory** (p. 32) with the information it needs to automatically call the **FooTypeSupport_register_type** (p. 221) 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 Connex infrastructure based on the type description provided in the XML files. If the **DDS_TypeSupport** (p. 214) has been registered with the **DDS_DomainParticipantFactory** (p. 32) this definition takes precedence over the description of the type given in the XML file.

The **DDS_DomainParticipantFactory_register_type_support** (p. 62) operation receives a **FooTypeSupport_register_type** (p. 221) function as a parameter. This function is normally generated using rtdsgen from a description of the corresponding 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 rtdsgen 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 **DDS_DomainParticipantFactory_register_type_support** (p. 62) passing the function that was generated by rtdsgen. E.g. FooTypeSupport::register_type 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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>register_type_fcn</i>	<< <i>in</i> >> (p. 1013) DDS_DomainParticipantFactory_RegisterTypeFunction (p. 33) to be used for registering the type with a DDS_DomainParticipant (p. 77).
<i>type_name</i>	<< <i>in</i> >> (p. 1013) Name the type is registered with.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

TypeSupport

5.3.4.45 DDS_DomainParticipantFactory_get_participants()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_get_participants (
    DDS_DomainParticipantFactory * self,
    struct DDS_DomainParticipantSeq * participants )
```

<<*extension*>> (p. 1012) 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. 1222).

DomainParticipants are ordered in the output sequence from the most recently created to the oldest.

MT Safety:

Safe.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>participants</i>	<< <i>inout</i> >> (p. 1013) a DomainParticipantSeq object where the set or list of participants will be returned

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

5.3.4.46 DDS_DomainParticipantFactory_set_thread_factory()

```
DDS_ReturnCode_t DDS_DomainParticipantFactory_set_thread_factory (
    DDS_DomainParticipantFactory * self,
    const struct DDS_ThreadFactory * thread_factory )
```

<<extension>> (p. 1012) Sets a **DDS_ThreadFactory** (p. 2027) in the **DDS_DomainParticipantFactory** (p. 32) that will be used to create the internal threads of the DDS middleware.

DDS threads are managed by the **DDS_DomainParticipant** (p. 77), which is responsible for creating and deleting threads throughout its lifecycle. Some threads are created when the **DDS_DomainParticipant** (p. 77) gets enabled (i.e., database, event, etc.) and others when special features are enabled (i.e., asynchronous publication thread).

DDS threads have **DDS_DomainParticipant** (p. 77) scope. This means every DomainParticipant creates all the necessary threads to operate and each **DDS_DomainParticipant** (p. 77) will create its own independent set of threads.

Each **DDS_DomainParticipant** (p. 77) creates an independent set of DDS threads and will use the **DDS_ThreadFactory** (p. 2027) that is held by the **DDS_DomainParticipantFactory** (p. 32) at the time the DomainParticipant is created. A **DDS_ThreadFactory** (p. 2027) is immutable from a **DDS_DomainParticipant** (p. 77) point of view. This means that a **DDS_DomainParticipant** (p. 77) will use throughout its lifecycle the same **DDS_ThreadFactory** (p. 2027) that was set when it was created, even if a new **DDS_ThreadFactory** (p. 2027) is set in the **DDS_DomainParticipantFactory** (p. 32) later. That is, if a new **DDS_ThreadFactory** (p. 2027) is set, a previously created **DDS_DomainParticipant** (p. 77) will not be affected.

ThreadFactory lifecycle: A **DDS_ThreadFactory** (p. 2027) instance must be alive while there are DomainParticipants using it. A **DDS_ThreadFactory** (p. 2027) instance can be deleted only after all DomainParticipants using it are deleted.

By default, the **DDS_DomainParticipantFactory** (p. 32) has an internal **DDS_ThreadFactory** (p. 2027) 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 **DDS_DomainParticipant** (p. 77) can occur concurrently with setting a **DDS_ThreadFactory** (p. 2027).

Parameters

<i>self</i>	<<in>> (p. 1013) Cannot be NULL.
<i>thread_factory</i>	<<in>> (p. 1013) Instance of a DDS_ThreadFactory (p. 2027) to be used for creating the DDS threads. If null is specified, the current DDS_ThreadFactory (p. 2027) is removed and the internal default will be used.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_ThreadFactory (p. 2027)

5.3.5 Variable Documentation

5.3.5.1 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 **DDS_DomainParticipantFactory_create_participant** (p. 41), this special value is used to indicate that the **DDS_DomainParticipant** (p. 77) should be created with the default **DDS_DomainParticipant** (p. 77) QoS by means of the operation **DDS_DomainParticipantFactory_get_default_participant_qos()** (p. 41) and using the resulting QoS to create the **DDS_DomainParticipant** (p. 77).

When used in **DDS_DomainParticipantFactory_set_default_participant_qos** (p. 39), 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 **DDS_DomainParticipantFactory_set_default_participant_qos** (p. 39) operation had never been called.

When used in **DDS_DomainParticipant_set_qos** (p. 151), this special value is used to indicate that the QoS of the **DDS_DomainParticipant** (p. 77) should be changed to match the current default QoS set in the **DDS_DomainParticipantFactory** (p. 32) that the **DDS_DomainParticipant** (p. 77) 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 **DDS_DomainParticipantFactory_get_default_participant_qos** (p. 41).

See also

NDDS_DISCOVERY_PEERS (p. 1352)

DDS_DomainParticipantFactory_create_participant() (p. 41)

DDS_DomainParticipantFactory_set_default_participant_qos() (p. 39)

DDS_DomainParticipant_set_qos() (p. 151)

Examples

HelloWorld_publisher.c, and **HelloWorld_subscriber.c**.

5.3.5.2 DDS_PARTICIPANT_CONFIG_PARAMS_DEFAULT

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

Special value for creating a **DDS_DomainParticipant** (p. 77) from configuration using default parameters.

This value can be used only in **DDS_DomainParticipantFactory_create_participant_from_config_w_params** (p. 60) and indicates that the **DDS_DomainParticipant** (p. 77) 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. 1719)

DDS_DomainParticipantFactory_create_participant_from_config_w_params (p. 60)

5.4 DomainParticipants

DDS_DomainParticipant (p. 77) entity and associated elements

Data Structures

- struct **DDS_InvalidLocalAdvanceNoticeStatus**
DDS_INVALID_LOCAL_ADVANCE_NOTICE_STATUS (p. 1232)
- struct **DDS_DomainParticipantListener**
 <<*interface*>> (p. 1013) *Listener for participant status.*
- struct **DDS_DomainParticipantQos**
*QoS policies supported by a **DDS_DomainParticipant** (p. 77) entity.*
- struct **DDS_DomainParticipantProtocolStatus**
 <<*extension*>> (p. 1012) *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.*
- struct **DDS_DomainParticipantSeq**
*Declares IDL sequence < **DDS_DomainParticipant** (p. 77) > .*

Macros

- #define **DDS_INVALID_LOCAL_ADVANCE_NOTICE_STATUS_EXPIRATION_MESSAGE_INITIALIZER**
*Initializer for new **DDS_InvalidLocalAdvanceNoticeStatus** (p. 1810).*
- #define **DDS_DomainParticipantListener_INITIALIZER**
*Initializer for new **DDS_DomainParticipantListener** (p. 1723).*
- #define **DDS_DomainParticipantQos_INITIALIZER**
Initializer for new QoS instances.
- #define **DDS_DomainParticipantProtocolStatus_INITIALIZER**
Initializer for new status instances.

Typedefs

- typedef DDS_DOMAINID_TYPE_NATIVE **DDS_DomainId_t**
*An integer that indicates in which domain a **DDS_DomainParticipant** (p. 77) communicates.*
- typedef void(* **DDS_DomainParticipantListener_InvalidLocalAdvanceNoticeStatusCallback**) (void *listener_data, **DDS_DomainParticipant** *participant, const struct **DDS_InvalidLocalAdvanceNoticeStatus** *invalid_local_advance_notice_status)
*Notifies the user that the identity or permissions of the **DDS_DomainParticipant** (p. 77) is about to expire.*
- typedef struct DDS_DomainParticipantImpl **DDS_DomainParticipant**
*<<interface>> (p. 1013) Container for all **DDS_DomainEntity** (p. 1363) objects.*

Functions

- **DDS_Boolean** **DDS_InvalidLocalAdvanceNoticeStatus_equals** (const struct **DDS_InvalidLocalAdvanceNoticeStatus** *left, const struct **DDS_InvalidLocalAdvanceNoticeStatus** *right)
*Compares two **DDS_InvalidLocalAdvanceNoticeStatus** (p. 1810) for equality.*
- **DDS_Boolean** **DDS_DomainParticipantQos_equals** (const struct **DDS_DomainParticipantQos** *self, const struct **DDS_DomainParticipantQos** *other)
*Compares two **DDS_DomainParticipantQos** (p. 1726) for equality.*
- **DDS_ReturnCode_t** **DDS_DomainParticipantQos_print** (const struct **DDS_DomainParticipantQos** *self)
*Prints this **DDS_DomainParticipantQos** (p. 1726) to stdout.*
- **DDS_ReturnCode_t** **DDS_DomainParticipantQos_to_string** (const struct **DDS_DomainParticipantQos** *self, char *string, **DDS_UnsignedLong** *string_size)
*Obtains a string representation of this **DDS_DomainParticipantQos** (p. 1726).*
- **DDS_ReturnCode_t** **DDS_DomainParticipantQos_to_string_w_params** (const struct **DDS_DomainParticipantQos** *self, char *string, **DDS_UnsignedLong** *string_size, const struct **DDS_DomainParticipantQos** *base, const struct **DDS_QosPrintFormat** *format)
*Obtains a string representation of this **DDS_DomainParticipantQos** (p. 1726).*
- **DDS_ReturnCode_t** **DDS_DomainParticipantQos_initialize** (struct **DDS_DomainParticipantQos** *self)
Initializer for new QoS instances.
- **DDS_ReturnCode_t** **DDS_DomainParticipantQos_finalize** (struct **DDS_DomainParticipantQos** *self)
*Free any dynamic memory allocated by the policies in this **DDS_DomainParticipantQos** (p. 1726).*
- **DDS_ReturnCode_t** **DDS_DomainParticipantQos_copy** (struct **DDS_DomainParticipantQos** *self, const struct **DDS_DomainParticipantQos** *source)
Copy the contents of the given QoS into this QoS.
- **DDS_Entity *** **DDS_DomainParticipant_as_entity** (**DDS_DomainParticipant** *domain)
*Access a **DDS_DomainParticipant** (p. 77)'s supertype instance.*
- **DDS_ReturnCode_t** **DDS_DomainParticipantProtocolStatus_initialize** (struct **DDS_DomainParticipantProtocolStatus** *self)
Initializer for new status instances.
- **DDS_ReturnCode_t** **DDS_DomainParticipantProtocolStatus_copy** (struct **DDS_DomainParticipantProtocolStatus** *self, const struct **DDS_DomainParticipantProtocolStatus** *source)
Copy the contents of the given status into this status.
- **DDS_ReturnCode_t** **DDS_DomainParticipantProtocolStatus_finalize** (struct **DDS_DomainParticipantProtocolStatus** *self)
Free any dynamic memory allocated by status instances.
- **DDS_Boolean** **DDS_DomainParticipantProtocolStatus_equals** (const struct **DDS_DomainParticipantProtocolStatus** *left, const struct **DDS_DomainParticipantProtocolStatus** *right)

Compares two **DDS_DomainParticipantProtocolStatus** (p. 1725) for equality.

- **DDS_ReturnCode_t DDS_DomainParticipant_get_default_topic_qos** (**DDS_DomainParticipant** *self, struct **DDS_TopicQos** *qos)

Copies the default **DDS_TopicQos** (p. 2042) values for this domain participant into the given **DDS_TopicQos** (p. 2042) instance.

- **DDS_ReturnCode_t DDS_DomainParticipant_set_default_topic_qos** (**DDS_DomainParticipant** *self, const struct **DDS_TopicQos** *qos)

Set the default **DDS_TopicQos** (p. 2042) values for this domain participant.

- **DDS_ReturnCode_t DDS_DomainParticipant_set_default_topic_qos_with_profile** (**DDS_DomainParticipant** *self, const char *library_name, const char *profile_name)

<<extension>> (p. 1012) Set the default **DDS_TopicQos** (p. 2042) values for this domain participant based on the input XML QoS profile.

- **DDS_ReturnCode_t DDS_DomainParticipant_get_default_publisher_qos** (**DDS_DomainParticipant** *self, struct **DDS_PublisherQos** *qos)

Copy the default **DDS_PublisherQos** (p. 1918) values into the provided **DDS_PublisherQos** (p. 1918) instance.

- **DDS_ReturnCode_t DDS_DomainParticipant_set_default_publisher_qos** (**DDS_DomainParticipant** *self, const struct **DDS_PublisherQos** *qos)

Set the default **DDS_PublisherQos** (p. 1918) values for this DomainParticipant.

- **DDS_ReturnCode_t DDS_DomainParticipant_set_default_publisher_qos_with_profile** (**DDS_DomainParticipant** *self, const char *library_name, const char *profile_name)

<<extension>> (p. 1012) Set the default **DDS_PublisherQos** (p. 1918) values for this DomainParticipant based on the input XML QoS profile.

- **DDS_ReturnCode_t DDS_DomainParticipant_get_default_datawriter_qos** (**DDS_DomainParticipant** *self, struct **DDS_DataWriterQos** *qos)

<<extension>> (p. 1012) Copy the default **DDS_DataWriterQos** (p. 1666) values into the provided **DDS_DataWriterQos** (p. 1666) instance.

- **DDS_ReturnCode_t DDS_DomainParticipant_set_default_datawriter_qos** (**DDS_DomainParticipant** *self, const struct **DDS_DataWriterQos** *qos)

<<extension>> (p. 1012) Set the default DataWriterQos values for this DomainParticipant.

- **DDS_ReturnCode_t DDS_DomainParticipant_set_default_datawriter_qos_with_profile** (**DDS_DomainParticipant** *self, const char *library_name, const char *profile_name)

<<extension>> (p. 1012) Set the default **DDS_DataWriterQos** (p. 1666) values for this domain participant based on the input XML QoS profile.

- **DDS_ReturnCode_t DDS_DomainParticipant_get_default_subscriber_qos** (**DDS_DomainParticipant** *self, struct **DDS_SubscriberQos** *qos)

Copy the default **DDS_SubscriberQos** (p. 2010) values into the provided **DDS_SubscriberQos** (p. 2010) instance.

- **DDS_ReturnCode_t DDS_DomainParticipant_set_default_subscriber_qos** (**DDS_DomainParticipant** *self, const struct **DDS_SubscriberQos** *qos)

Set the default **DDS_SubscriberQos** (p. 2010) values for this DomainParticipant.

- **DDS_ReturnCode_t DDS_DomainParticipant_set_default_subscriber_qos_with_profile** (**DDS_DomainParticipant** *self, const char *library_name, const char *profile_name)

<<extension>> (p. 1012) Set the default **DDS_SubscriberQos** (p. 2010) values for this DomainParticipant based on the input XML QoS profile.

- **DDS_ReturnCode_t DDS_DomainParticipant_get_default_datareader_qos** (**DDS_DomainParticipant** *self, struct **DDS_DataReaderQos** *qos)

<<extension>> (p. 1012) Copy the default **DDS_DataReaderQos** (p. 1611) values into the provided **DDS_DataReaderQos** (p. 1611) instance.

- **DDS_ReturnCode_t DDS_DomainParticipant_set_default_datareader_qos** (**DDS_DomainParticipant** *self, const struct **DDS_DataReaderQos** *qos)

<<extension>> (p. 1012) Set the default **DDS_DataReaderQos** (p. 1611) values for this domain participant.

- **DDS_ReturnCode_t DDS_DomainParticipant_set_default_datareader_qos_with_profile** (DDS_DomainParticipant *self, const char *library_name, const char *profile_name)
 <<extension>> (p. 1012) Set the default **DDS_DataReaderQos** (p. 1611) values for this DomainParticipant based on the input XML QoS profile.
- **DDS_ReturnCode_t DDS_DomainParticipant_get_default_flowcontroller_property** (DDS_DomainParticipant *self, struct DDS_FlowControllerProperty_t *prop)
 <<extension>> (p. 1012) Copies the default **DDS_FlowControllerProperty_t** (p. 1799) values for this domain participant into the given **DDS_FlowControllerProperty_t** (p. 1799) instance.
- **DDS_ReturnCode_t DDS_DomainParticipant_set_default_flowcontroller_property** (DDS_DomainParticipant *self, const struct DDS_FlowControllerProperty_t *prop)
 <<extension>> (p. 1012) Set the default **DDS_FlowControllerProperty_t** (p. 1799) values for this domain participant.
- **const char * DDS_DomainParticipant_get_default_library** (DDS_DomainParticipant *self)
 <<extension>> (p. 1012) Gets the default XML library associated with a **DDS_DomainParticipant** (p. 77).
- **const char * DDS_DomainParticipant_get_default_profile** (DDS_DomainParticipant *self)
 <<extension>> (p. 1012) Gets the default XML profile associated with a **DDS_DomainParticipant** (p. 77).
- **const char * DDS_DomainParticipant_get_default_profile_library** (DDS_DomainParticipant *self)
 <<extension>> (p. 1012) Gets the library where the default XML QoS profile is contained for a **DDS_DomainParticipant** (p. 77).
- **DDS_ReturnCode_t DDS_DomainParticipant_set_default_library** (DDS_DomainParticipant *self, const char *library_name)
 <<extension>> (p. 1012) Sets the default XML library for a **DDS_DomainParticipant** (p. 77).
- **DDS_ReturnCode_t DDS_DomainParticipant_set_default_profile** (DDS_DomainParticipant *self, const char *library_name, const char *profile_name)
 <<extension>> (p. 1012) Sets the default XML profile for a **DDS_DomainParticipant** (p. 77).
- **DDS_Publisher * DDS_DomainParticipant_create_publisher** (DDS_DomainParticipant *self, const struct DDS_PublisherQos *qos, const struct DDS_PublisherListener *listener, DDS_StatusMask mask)
 Creates a **DDS_Publisher** (p. 443) with the desired QoS policies and attaches to it the specified **DDS_PublisherListener** (p. 1917).
- **DDS_Publisher * DDS_DomainParticipant_create_publisher_with_profile** (DDS_DomainParticipant *self, const char *library_name, const char *profile_name, const struct DDS_PublisherListener *listener, DDS_StatusMask mask)
 <<extension>> (p. 1012) Creates a new **DDS_Publisher** (p. 443) object using the **DDS_PublisherQos** (p. 1918) associated with the input XML QoS profile.
- **DDS_ReturnCode_t DDS_DomainParticipant_delete_publisher** (DDS_DomainParticipant *self, DDS_Publisher *p)
 Deletes an existing **DDS_Publisher** (p. 443).
- **DDS_Subscriber * DDS_DomainParticipant_create_subscriber** (DDS_DomainParticipant *self, const struct DDS_SubscriberQos *qos, const struct DDS_SubscriberListener *listener, DDS_StatusMask mask)
 Creates a **DDS_Subscriber** (p. 572) with the desired QoS policies and attaches to it the specified **DDS_SubscriberListener** (p. 2009).
- **DDS_Subscriber * DDS_DomainParticipant_create_subscriber_with_profile** (DDS_DomainParticipant *self, const char *library_name, const char *profile_name, const struct DDS_SubscriberListener *listener, DDS_StatusMask mask)
 <<extension>> (p. 1012) Creates a new **DDS_Subscriber** (p. 572) object using the **DDS_PublisherQos** (p. 1918) associated with the input XML QoS profile.
- **DDS_ReturnCode_t DDS_DomainParticipant_delete_subscriber** (DDS_DomainParticipant *self, DDS_Subscriber *s)
 Deletes an existing **DDS_Subscriber** (p. 572).
- **DDS_DataWriter * DDS_DomainParticipant_create_datawriter** (DDS_DomainParticipant *self, DDS_Topic *topic, const struct DDS_DataWriterQos *qos, const struct DDS_DataWriterListener *listener, DDS_StatusMask mask)

- <<extension>>** (p. 1012) Creates a **DDS_DataWriter** (p. 485) that will be attached and belong to the implicit **DDS_Publisher** (p. 443).
- **DDS_DataWriter * DDS_DomainParticipant_create_datawriter_with_profile** (**DDS_DomainParticipant** *self, **DDS_Topic** *topic, const char *library_name, const char *profile_name, const struct **DDS_DataWriterListener** *listener, **DDS_StatusMask** mask)
<<extension>> (p. 1012) Creates a **DDS_DataWriter** (p. 485) using a XML QoS profile that will be attached and belong to the implicit **DDS_Publisher** (p. 443).
 - **DDS_ReturnCode_t DDS_DomainParticipant_delete_datawriter** (**DDS_DomainParticipant** *self, **DDS_DataWriter** *a_datawriter)
<<extension>> (p. 1012) Deletes a **DDS_DataWriter** (p. 485) that belongs to the implicit **DDS_Publisher** (p. 443).
 - **DDS_DataReader * DDS_DomainParticipant_create_datareader** (**DDS_DomainParticipant** *self, **DDS_TopicDescription** *topic, const struct **DDS_DataReaderQos** *qos, const struct **DDS_DataReaderListener** *listener, **DDS_StatusMask** mask)
<<extension>> (p. 1012) Creates a **DDS_DataReader** (p. 615) that will be attached and belong to the implicit **DDS_Subscriber** (p. 572).
 - **DDS_DataReader * DDS_DomainParticipant_create_datareader_with_profile** (**DDS_DomainParticipant** *self, **DDS_TopicDescription** *topic, const char *library_name, const char *profile_name, const struct **DDS_DataReaderListener** *listener, **DDS_StatusMask** mask)
<<extension>> (p. 1012) Creates a **DDS_DataReader** (p. 615) using a XML QoS profile that will be attached and belong to the implicit **DDS_Subscriber** (p. 572).
 - **DDS_ReturnCode_t DDS_DomainParticipant_delete_datareader** (**DDS_DomainParticipant** *self, **DDS_DataReader** *a_datareader)
<<extension>> (p. 1012) Deletes a **DDS_DataReader** (p. 615) that belongs to the implicit **DDS_Subscriber** (p. 572).
 - **DDS_Topic * DDS_DomainParticipant_create_topic** (**DDS_DomainParticipant** *self, const char *topic_name, const char *type_name, const struct **DDS_TopicQos** *qos, const struct **DDS_TopicListener** *listener, **DDS_StatusMask** mask)
Creates a **DDS_Topic** (p. 175) with the desired QoS policies and attaches to it the specified **DDS_TopicListener** (p. 2041).
 - **DDS_Topic * DDS_DomainParticipant_create_topic_with_profile** (**DDS_DomainParticipant** *self, const char *topic_name, const char *type_name, const char *library_name, const char *profile_name, const struct **DDS_TopicListener** *listener, **DDS_StatusMask** mask)
<<extension>> (p. 1012) Creates a new **DDS_Topic** (p. 175) object using the **DDS_PublisherQos** (p. 1918) associated with the input XML QoS profile.
 - **DDS_ReturnCode_t DDS_DomainParticipant_delete_topic** (**DDS_DomainParticipant** *self, **DDS_Topic** *topic)
Deletes a **DDS_Topic** (p. 175).
 - **DDS_ContentFilteredTopic * DDS_DomainParticipant_create_contentfilteredtopic** (**DDS_DomainParticipant** *self, const char *name, **DDS_Topic** *related_topic, const char *filter_expression, const struct **DDS_StringSeq** *expression_parameters)
Creates a **DDS_ContentFilteredTopic** (p. 177), that can be used to do content-based subscriptions.
 - **DDS_ContentFilteredTopic * DDS_DomainParticipant_create_contentfilteredtopic_with_filter** (**DDS_DomainParticipant** *self, const char *name, const **DDS_Topic** *related_topic, const char *filter_expression, const struct **DDS_StringSeq** *expression_parameters, const char *filter_name)
<<extension>> (p. 1012) Creates a **DDS_ContentFilteredTopic** (p. 177) using the specified filter to do content-based subscriptions.
 - **DDS_ReturnCode_t DDS_DomainParticipant_delete_contentfilteredtopic** (**DDS_DomainParticipant** *self, **DDS_ContentFilteredTopic** *a_contentfilteredtopic)
Deletes a **DDS_ContentFilteredTopic** (p. 177).
 - **DDS_ReturnCode_t DDS_DomainParticipant_register_contentfilter** (**DDS_DomainParticipant** *self, const char *filter_name, const struct **DDS_ContentFilter** *contentfilter)
<<extension>> (p. 1012) Register a content filter which can be used to create a **DDS_ContentFilteredTopic** (p. 177).

- `void * DDS_DomainParticipant_lookup_contentfilter ( DDS_DomainParticipant *self, const char *filter_name)`
<<extension>> (p. 1012) Lookup a content filter previously registered with DDS_DomainParticipant_register_contentfilter (p. 120).
- `DDS_ReturnCode_t DDS_DomainParticipant_unregister_contentfilter ( DDS_DomainParticipant *self, const char *filter_name)`
<<extension>> (p. 1012) Unregister a content filter previously registered with DDS_DomainParticipant_register_contentfilter (p. 120).
- `DDS_MultiTopic * DDS_DomainParticipant_create_multitopic ( DDS_DomainParticipant *self, const char *name, const char *type_name, const char *subscription_expression, const struct DDS_StringSeq *expression_parameters)`
[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.
- `DDS_ReturnCode_t DDS_DomainParticipant_delete_multitopic ( DDS_DomainParticipant *self, DDS_MultiTopic *a_multitopic)`
[Not supported (optional)] Deletes a DDS_MultiTopic (p. 185).
- `DDS_FlowController * DDS_DomainParticipant_create_flowcontroller ( DDS_DomainParticipant *self, const char *name, const struct DDS_FlowControllerProperty_t *prop)`
<<extension>> (p. 1012) Creates a DDS_FlowController (p. 559) with the desired property.
- `DDS_ReturnCode_t DDS_DomainParticipant_delete_flowcontroller ( DDS_DomainParticipant *self, DDS_FlowController *fc)`
<<extension>> (p. 1012) Deletes an existing DDS_FlowController (p. 559).
- `const DDS_TypeCode * DDS_DomainParticipant_get_typecode ( DDS_DomainParticipant *self, const char *type_name)`
<<extension>> (p. 1012) Retrieves the DDS_TypeCode (p. 2071) of a type registered with the DDS_DomainParticipant (p. 77) based on the registration name.
- `DDS_Topic * DDS_DomainParticipant_find_topic ( DDS_DomainParticipant *self, const char *topic_name, const struct DDS_Duration_t *timeout)`
Finds an existing (or ready to exist) DDS_Topic (p. 175), based on its name.
- `DDS_TopicDescription * DDS_DomainParticipant_lookup_topicdescription ( DDS_DomainParticipant *self, const char *topic_name)`
Looks up an existing, locally created DDS_TopicDescription (p. 175), based on its name.
- `DDS_FlowController * DDS_DomainParticipant_lookup_flowcontroller ( DDS_DomainParticipant *self, const char *name)`
<<extension>> (p. 1012) Looks up an existing locally-created DDS_FlowController (p. 559), based on its name.
- `DDS_ReturnCode_t DDS_DomainParticipant_get_participant_protocol_status ( DDS_DomainParticipant *self, struct DDS_DomainParticipantProtocolStatus *status)`
<<extension>> (p. 1012) Get the domain participant protocol status for this participant.
- `DDS_Subscriber * DDS_DomainParticipant_get_builtin_subscriber ( DDS_DomainParticipant *self)`
Accesses the built-in DDS_Subscriber (p. 572).
- `DDS_Publisher * DDS_DomainParticipant_get_implicit_publisher ( DDS_DomainParticipant *self)`
<<extension>> (p. 1012) Returns the implicit DDS_Publisher (p. 443). If an implicit Publisher does not already exist, this creates one.
- `DDS_Subscriber * DDS_DomainParticipant_get_implicit_subscriber ( DDS_DomainParticipant *self)`
<<extension>> (p. 1012) Returns the implicit DDS_Subscriber (p. 572). If an implicit Subscriber does not already exist, this creates one.
- `DDS_ReturnCode_t DDS_DomainParticipant_ignore_participant ( DDS_DomainParticipant *self, const DDS_InstanceHandle_t *handle)`
Instructs RTI Connex to locally ignore a remote DDS_DomainParticipant (p. 77).

- **DDS_ReturnCode_t DDS_DomainParticipant_ignore_topic** (DDS_DomainParticipant *self, const DDS_InstanceHandle_t *handle)
*Instructs RTI Connex to locally ignore a **DDS_Topic** (p. 175).*
- **DDS_ReturnCode_t DDS_DomainParticipant_ignore_publication** (DDS_DomainParticipant *self, const DDS_InstanceHandle_t *handle)
Instructs RTI Connex to locally ignore a publication.
- **DDS_ReturnCode_t DDS_DomainParticipant_ignore_subscription** (DDS_DomainParticipant *self, const DDS_InstanceHandle_t *handle)
Instructs RTI Connex to locally ignore a subscription.
- **DDS_ReturnCode_t DDS_DomainParticipant_banish_ignored_participants** (DDS_DomainParticipant *self)
*<<extension>> (p. 1012) Prevents ignored remote DomainParticipants from receiving traffic from the local **DDS_DomainParticipant** (p. 77).*
- **DDS_DomainId_t DDS_DomainParticipant_get_domain_id** (DDS_DomainParticipant *self)
Get the unique domain identifier.
- **DDS_ReturnCode_t DDS_DomainParticipant_delete_contained_entities** (DDS_DomainParticipant *self)
*Delete all the entities that were created by means of the "create" operations on the **DDS_DomainParticipant** (p. 77).*
- **DDS_ReturnCode_t DDS_DomainParticipant_assert_liveliness** (DDS_DomainParticipant *self)
*Manually asserts the liveliness of this **DDS_DomainParticipant** (p. 77).*
- **DDS_ReturnCode_t DDS_DomainParticipant_get_publishers** (DDS_DomainParticipant *self, struct DDS_PublisherSeq *publishers)
<<extension>> (p. 1012) Allows the application to access all the publishers the participant has.
- **DDS_ReturnCode_t DDS_DomainParticipant_get_subscribers** (DDS_DomainParticipant *self, struct DDS_SubscriberSeq *subscribers)
<<extension>> (p. 1012) Allows the application to access all the subscribers the participant has.
- **DDS_ReturnCode_t DDS_DomainParticipant_get_current_time** (DDS_DomainParticipant *self, struct DDS_Time_t *current_time)
Returns the current value of the time.
- **DDS_Boolean DDS_DomainParticipant_contains_entity** (DDS_DomainParticipant *self, const DDS_InstanceHandle_t *a_handle)
*Completes successfully with **DDS_BOOLEAN_TRUE** (p. 1201) if the referenced **DDS_Entity** (p. 1360) is contained by the **DDS_DomainParticipant** (p. 77).*
- **DDS_ReturnCode_t DDS_DomainParticipant_register_durable_subscription** (DDS_DomainParticipant *self, const struct DDS_EndpointGroup_t *group, const char *topic_name)
*<<extension>> (p. 1012) Registers a Durable Subscription on the specified **DDS_Topic** (p. 175) on all Persistence Services.*
- **DDS_ReturnCode_t DDS_DomainParticipant_delete_durable_subscription** (DDS_DomainParticipant *self, const struct DDS_EndpointGroup_t *group)
<<extension>> (p. 1012) Deletes an existing Durable Subscription on all Persistence Services.
- **DDS_ReturnCode_t DDS_DomainParticipant_resume_endpoint_discovery** (DDS_DomainParticipant *self, const DDS_InstanceHandle_t *remote_participant_handle)
*<<extension>> (p. 1012) Initiates endpoint discovery with the specified remote **DDS_DomainParticipant** (p. 77).*
- **DDS_ReturnCode_t DDS_DomainParticipant_set_dns_tracker_polling_period** (DDS_DomainParticipant *self, const struct DDS_Duration_t *polling_period)
<<extension>> (p. 1012) Configures the frequency in which the DNS tracker queries the DNS service.
- **DDS_ReturnCode_t DDS_DomainParticipant_get_dns_tracker_polling_period** (DDS_DomainParticipant *self, struct DDS_Duration_t *polling_period)
<<extension>> (p. 1012) Retrieves the frequency used by the DNS tracker thread to query the DNS service.
- **DDS_ReturnCode_t DDS_DomainParticipant_get_discovered_participants** (DDS_DomainParticipant *self, struct DDS_InstanceHandleSeq *participant_handles)

Returns a list of discovered **DDS_DomainParticipant** (p. 77) entities.

- **DDS_ReturnCode_t DDS_DomainParticipant_get_discovered_participants_from_subject_name** (**DDS_↵_DomainParticipant** *self, struct **DDS_InstanceHandleSeq** *participant_handles, const char *subject_name)
<<extension>> (p. 1012) Returns a list of discovered **DDS_DomainParticipant** (p. 77) entities that have the given **DDS_EntityNameQosPolicy::name** (p. 1790).
- **DDS_ReturnCode_t DDS_DomainParticipant_get_discovered_participant_data** (**DDS_Domain_↵Participant** *self, struct **DDS_ParticipantBuiltinTopicData** *participant_data, const **DDS_InstanceHandle_t** *participant_handle)
Returns **DDS_ParticipantBuiltinTopicData** (p. 1870) for the specified **DDS_DomainParticipant** (p. 77).
- **DDS_ReturnCode_t DDS_DomainParticipant_get_discovered_participant_subject_name** (**DDS_↵DomainParticipant** *self, char *subject_name, **DDS_UnsignedLong** *subject_name_size, const **DDS_↵InstanceHandle_t** *participant_handle)
<<extension>> (p. 1012) Returns **DDS_EntityNameQosPolicy::name** (p. 1790) for the specified **DDS_Domain_↵Participant** (p. 77).
- **DDS_ReturnCode_t DDS_DomainParticipant_get_discovered_topics** (**DDS_DomainParticipant** *self, struct **DDS_InstanceHandleSeq** *topic_handles)
Returns list of discovered **DDS_Topic** (p. 175) objects.
- **DDS_ReturnCode_t DDS_DomainParticipant_get_discovered_topic_data** (**DDS_DomainParticipant** *self, struct **DDS_TopicBuiltinTopicData** *topic_data, const **DDS_InstanceHandle_t** *topic_handle)
Returns **DDS_TopicBuiltinTopicData** (p. 2034) for the specified **DDS_Topic** (p. 175).
- **DDS_ReturnCode_t DDS_DomainParticipant_take_discovery_snapshot** (**DDS_DomainParticipant** *self, const char *file_name)
Take a snapshot of the remote participants discovered by a local one.
- **DDS_ReturnCode_t DDS_DomainParticipant_add_peer** (**DDS_DomainParticipant** *self, const char *peer_desc_string)
<<extension>> (p. 1012) Attempt to contact one or more additional peer participants.
- **DDS_ReturnCode_t DDS_DomainParticipant_remove_peer** (**DDS_DomainParticipant** *self, const char *peer_desc_string)
<<extension>> (p. 1012) Remove one or more peer participants from the list of peers with which this **DDS_Domain_↵Participant** (p. 77) will try to communicate.
- **DDS_ReturnCode_t DDS_DomainParticipant_set_qos** (**DDS_DomainParticipant** *self, const struct **DDS_↵DomainParticipantQos** *qos)
Change the QoS of this **DomainParticipant**.
- **DDS_ReturnCode_t DDS_DomainParticipant_set_qos_with_profile** (**DDS_DomainParticipant** *self, const char *library_name, const char *profile_name)
<<extension>> (p. 1012) Change the QoS of this domain participant using the input XML QoS profile.
- **DDS_ReturnCode_t DDS_DomainParticipant_get_qos** (**DDS_DomainParticipant** *self, struct **DDS_↵DomainParticipantQos** *qos)
Get the participant QoS.
- **DDS_ReturnCode_t DDS_DomainParticipant_set_property** (**DDS_DomainParticipant** *self, const char *property_name, const char *value, **DDS_Boolean** propagate)
Set the value for a property that applies to a **DomainParticipant**.
- **DDS_ReturnCode_t DDS_DomainParticipant_set_listener** (**DDS_DomainParticipant** *self, const struct **DDS_DomainParticipantListener** *l, **DDS_StatusMask** mask)
Sets the participant listener.
- struct **DDS_DomainParticipantListener** **DDS_DomainParticipant_get_listener** (**DDS_DomainParticipant** *self)
Get the participant listener.
- **DDS_ReturnCode_t DDS_DomainParticipant_get_listenerX** (**DDS_DomainParticipant** *self, struct **DDS_↵DomainParticipantListener** *listener)

- <<*extension*>> (p. 1012) Get the participant listener.
- **DDS_Publisher * DDS_DomainParticipant_lookup_publisher_by_name** (DDS_DomainParticipant *self, const char *publisher_name)
 - <<*extension*>> (p. 1012) Looks up a **DDS_Publisher** (p. 443) by its entity name within this **DDS_DomainParticipant** (p. 77).
- **DDS_Subscriber * DDS_DomainParticipant_lookup_subscriber_by_name** (DDS_DomainParticipant *self, const char *subscriber_name)
 - <<*extension*>> (p. 1012) Retrieves a **DDS_Subscriber** (p. 572) by its entity name within this **DDS_DomainParticipant** (p. 77).
- **DDS_DataWriter * DDS_DomainParticipant_lookup_datawriter_by_name** (DDS_DomainParticipant *self, const char *datawriter_full_name)
 - <<*extension*>> (p. 1012) Looks up a **DDS_DataWriter** (p. 485) by its entity name within this **DDS_DomainParticipant** (p. 77).
- **DDS_DataReader * DDS_DomainParticipant_lookup_datareader_by_name** (DDS_DomainParticipant *self, const char *datareader_full_name)
 - <<*extension*>> (p. 1012) Retrieves up a **DDS_DataReader** (p. 615) by its entity name in this **DDS_DomainParticipant** (p. 77).

Variables

- const struct **DDS_DomainParticipantFactoryQos * DDS_DOMAINPARTICIPANTFACTORY_QOS_PRINT_↔_ALL**
 - Special value which can be supplied to **DDS_DomainParticipantFactoryQos_to_string_w_params** (p. 35) indicating that all of the QoS should be printed.
- const struct **DDS_DomainParticipantQos * DDS_DOMAINPARTICIPANT_QOS_PRINT_ALL**
 - Special value which can be supplied to **DDS_DomainParticipantQos_to_string_w_params** (p. 81) indicating that all of the QoS should be printed.
- const struct **DDS_TopicQos DDS_TOPIC_QOS_DEFAULT**
 - Special value for creating a **DDS_Topic** (p. 175) with default QoS.
- const struct **DDS_PublisherQos DDS_PUBLISHER_QOS_DEFAULT**
 - Special value for creating a **DDS_Publisher** (p. 443) with default QoS.
- const struct **DDS_SubscriberQos DDS_SUBSCRIBER_QOS_DEFAULT**
 - Special value for creating a **DDS_Subscriber** (p. 572) with default QoS.
- const struct **DDS_FlowControllerProperty_t DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT**
 - <<*extension*>> (p. 1012) Special value for creating a **DDS_FlowController** (p. 559) with default property.
- const struct **DDS_PublisherQos * DDS_PUBLISHER_QOS_PRINT_ALL**
 - Special value which can be supplied to **DDS_PublisherQos_to_string_w_params** (p. 446) indicating that all of the QoS should be printed.
- const char *const **DDS_SQLFILTER_NAME**
 - <<*extension*>> (p. 1012) The name of the built-in SQL filter that can be used with ContentFilteredTopics and Multi↔Channel DataWriters.
- const char *const **DDS_STRINGMATCHFILTER_NAME**
 - <<*extension*>> (p. 1012) The name of the built-in StringMatch filter that can be used with ContentFilteredTopics and MultiChannel DataWriters.
- const struct **DDS_SubscriberQos * DDS_SUBSCRIBER_QOS_PRINT_ALL**
 - Special value which can be supplied to **DDS_SubscriberQos_to_string_w_params** (p. 576) indicating that all of the QoS should be printed.
- const struct **DDS_TopicQos * DDS_TOPIC_QOS_PRINT_ALL**
 - Special value which can be supplied to **DDS_TopicQos_to_string_w_params** (p. 190) indicating that all of the QoS should be printed.

5.4.1 Detailed Description

DDS_DomainParticipant (p. 77) entity and associated elements

5.4.2 Macro Definition Documentation

5.4.2.1 DDS_INVALID_LOCAL_ADVANCE_NOTICE_STATUS_EXPIRATION_MESSAGE_INITIALIZER

```
#define DDS_INVALID_LOCAL_ADVANCE_NOTICE_STATUS_EXPIRATION_MESSAGE_INITIALIZER
```

Initializer for new **DDS_InvalidLocalAdvanceNoticeStatus** (p. 1810).

5.4.2.2 DDS_DomainParticipantListener_INITIALIZER

```
#define DDS_DomainParticipantListener_INITIALIZER
```

Initializer for new **DDS_DomainParticipantListener** (p. 1723).

No memory is allocated. New **DDS_DomainParticipantListener** (p. 1723) instances stored in the stack should be initialized with this value before they are passed to any functions.

```
struct DDS_DomainParticipantListener listener = DDS_DomainParticipantListener_INITIALIZER;
```

```
/* initialize listener functions */
```

```
listener.as_subscriberlistener.as_datareaderlistener.on_data_available = ....;
```

```
DDS_DomainParticipant_set_listener(myParticipant, &listener, mask);
```

See also

DDS_DomainParticipant_set_listener (p. 154)

DDS_DomainParticipantListener (p. 1723)

5.4.2.3 DDS_DomainParticipantQos_INITIALIZER

```
#define DDS_DomainParticipantQos_INITIALIZER
```

Initializer for new QoS instances.

New **DDS_DomainParticipantQos** (p. 1726) instances stored on the stack should be initialized with this value before they are passed to any functions. This step ensures that those contained QoS policies that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a new QoS structure is to initialize it on the stack at the time of its creation:

```
struct DDS_DomainParticipantQos myQos = DDS_DomainParticipantQos_INITIALIZER;
```

Note that the above assignment is not a substitute for calling **DDS_DomainParticipant_get_qos** (p. 152) or **DDS_DomainParticipantFactory_get_default_participant_qos** (p. 41); one of those functions should be called subsequently to setting the QoS of any new or existing entity. **DDS_DomainParticipantQos_finalize** (p. 82) should be called to free the contained QoS policies that use dynamic memory:

```
struct DDS_DomainParticipantQos myQos = DDS_DomainParticipantQos_INITIALIZER;
```

```
DDS_DomainParticipantFactory_get_default_participant_qos(myFactory, &myQos);
```

```
DDS_DomainParticipant_set_qos(myParticipant, &myQos);
```

```
DDS_DomainParticipantQos_finalize(&myQos);
```

See also

DDS_DomainParticipantFactory_get_default_participant_qos (p. 41)

DDS_DomainParticipantQos_finalize (p. 82)

5.4.2.4 DDS_DomainParticipantProtocolStatus_INITIALIZER

```
#define DDS_DomainParticipantProtocolStatus_INITIALIZER
```

Initializer for new status instances.

New **DDS_DomainParticipantProtocolStatus** (p. 1725) instances stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those fields that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a status structure is to initialize it on the stack at the time of its creation. **DDS_DomainParticipantProtocolStatus_finalize** (p. 85) should be called to free the contained fields that use dynamic memory:

```
struct DDS_DomainParticipantProtocolStatus myStatus = DDS_DomainParticipantProtocolStatus_INITIALIZER;
```

```
DDS_DomainParticipant_get_participant_protocol_status(myParticipant, &myStatus);
```

```
DDS_DomainParticipantProtocolStatus_finalize(&myStatus);
```

See also

DDS_DomainParticipant_get_participant_protocol_status (p. 128)

DDS_DomainParticipantProtocolStatus_finalize (p. 85)

5.4.3 Typedef Documentation

5.4.3.1 DDS_DomainId_t

```
typedef DDS_DOMAINID_TYPE_NATIVE DDS_DomainId_t
```

An integer that indicates in which domain a **DDS_DomainParticipant** (p. 77) communicates.

Participants with the same **DDS_DomainId_t** (p. 77) 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. 1978) (specifically **DDS_RtpsWellKnownPorts_t::domain_id_gain** (p. 1981))

5.4.3.2 DDS_DomainParticipantListener_InvalidLocalAdvanceNoticeStatusCallback

```
typedef void(* DDS_DomainParticipantListener_InvalidLocalAdvanceNoticeStatusCallback) (void *listener↔_data, DDS_DomainParticipant *participant, const struct DDS_InvalidLocalAdvanceNoticeStatus *invalid_local_advance_notice_status)
```

Notifies the user that the identity or permissions of the **DDS_DomainParticipant** (p. 77) is about to expire.

5.4.3.3 DDS_DomainParticipant

```
typedef struct DDS_DomainParticipantImpl DDS_DomainParticipant
```

<<*interface*>> (p. 1013) Container for all **DDS_DomainEntity** (p. 1363) objects.

The DomainParticipant object plays several roles:

- It acts as a container for all other **DDS_Entity** (p. 1360) objects.
- It acts as a *factory* for the **DDS_Publisher** (p. 443), **DDS_Subscriber** (p. 572), **DDS_Topic** (p. 175) and **DDS_↔MultiTopic** (p. 185) **DDS_Entity** (p. 1360) 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 (**DDS_DomainParticipant_ignore_participant** (p. 131)), publication (**DDS_DomainParticipant_ignore_publication** (p. 132)), subscription (**DDS_DomainParticipant_ignore_subscription()** (p. 133)) or topic (**DDS_DomainParticipant_ignore_topic()** (p. 132)).

The following operations may be called even if the **DDS_DomainParticipant** (p. 77) is not enabled. Operations NOT in this list will fail with **DDS_RETCODE_NOT_ENABLED** (p. 1222) \ if called on a disabled DomainParticipant.

- **DDS_Entity_enable** (p. 1363),
- **DDS_DomainParticipant_set_qos** (p. 151), **DDS_DomainParticipant_set_qos_with_profile** (p. 152), **DDS_DomainParticipant_get_qos** (p. 152),
- **DDS_DomainParticipant_set_listener** (p. 154), **DDS_DomainParticipant_get_listener** (p. 154),
- Factory operations: **DDS_DomainParticipant_create_flowcontroller** (p. 124), **DDS_DomainParticipant_create_topic** (p. 115), **DDS_DomainParticipant_create_topic_with_profile** (p. 116), **DDS_DomainParticipant_create_publisher** (p. 103), **DDS_DomainParticipant_create_publisher_with_profile** (p. 104), **DDS_DomainParticipant_create_subscriber** (p. 106), **DDS_DomainParticipant_create_subscriber_with_profile** (p. 107), **DDS_DomainParticipant_delete_flowcontroller** (p. 124), **DDS_DomainParticipant_delete_topic** (p. 117), **DDS_DomainParticipant_delete_publisher** (p. 105), **DDS_DomainParticipant_delete_subscriber** (p. 108), **DDS_DomainParticipant_set_default_topic_qos** (p. 87), **DDS_DomainParticipant_set_default_topic_qos_with_profile** (p. 88), **DDS_DomainParticipant_get_default_topic_qos** (p. 86), **DDS_DomainParticipant_set_default_publisher_qos** (p. 89), **DDS_DomainParticipant_set_default_publisher_qos_with_profile** (p. 90), **DDS_DomainParticipant_get_default_publisher_qos** (p. 89), **DDS_DomainParticipant_set_default_subscriber_qos** (p. 94), **DDS_DomainParticipant_set_default_subscriber_qos_with_profile** (p. 95), **DDS_DomainParticipant_get_default_subscriber_qos** (p. 94), **DDS_DomainParticipant_delete_contained_entities** (p. 135), **DDS_DomainParticipant_set_default_datareader_qos** (p. 97), **DDS_DomainParticipant_set_default_datareader_qos_with_profile** (p. 97), **DDS_DomainParticipant_get_default_datareader_qos** (p. 96), **DDS_DomainParticipant_set_default_datawriter_qos** (p. 92), **DDS_DomainParticipant_set_default_datawriter_qos_with_profile** (p. 93), **DDS_DomainParticipant_get_default_datawriter_qos** (p. 91), **DDS_DomainParticipant_set_default_library** (p. 101), **DDS_DomainParticipant_set_default_profile** (p. 102), **DDS_DomainParticipant_set_default_flowcontroller_property** (p. 99), **DDS_DomainParticipant_get_default_flowcontroller_property** (p. 98), ;
- Operations for looking up topics: **DDS_DomainParticipant_lookup_topicdescription** (p. 127);
- Operations that access status: **DDS_Entity_get_statuscondition** (p. 1365), **DDS_Entity_get_status_changes** (p. 1365).

QoS:

DDS_DomainParticipantQos (p. 1726)

Status:

Status Kinds (p. 1222)

Listener:

DDS_DomainParticipantListener (p. 1723)

See also

Operations Allowed in Listener Callbacks (p. 1818)

5.4.4 Function Documentation

5.4.4.1 DDS_InvalidLocalAdvanceNoticeStatus_equals()

```
DDS_Boolean DDS_InvalidLocalAdvanceNoticeStatus_equals (
    const struct DDS_InvalidLocalAdvanceNoticeStatus * left,
    const struct DDS_InvalidLocalAdvanceNoticeStatus * right )
```

Compares two **DDS_InvalidLocalAdvanceNoticeStatus** (p. 1810) for equality.

Parameters

<i>left</i>	<< <i>in</i> >> (p. 1013) This InvalidLocalAdvanceNoticeStatus. Can be NULL.
<i>right</i>	<< <i>in</i> >> (p. 1013) The other InvalidLocalAdvanceNoticeStatus to be compared with this status instance. Can be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two InvalidLocalAdvanceNoticeStatus have equal values, or **DDS_↵
BOOLEAN_FALSE** (p. 1201) otherwise.

5.4.4.2 DDS_DomainParticipantQos_equals()

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

Compares two **DDS_DomainParticipantQos** (p. 1726) for equality.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) This DomainParticipantQos.
<i>other</i>	<< <i>in</i> >> (p. 1013) The other DomainParticipantQos to be compared with this DomainParticipantQos.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two Qos objects are equal or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.4.4.3 DDS_DomainParticipantQos_print()

```
DDS_ReturnCode_t DDS_DomainParticipantQos_print (
    const struct DDS_DomainParticipantQos * self )
```

Prints this **DDS_DomainParticipantQos** (p. 1726) to stdout.

Only the differences between this **DDS_DomainParticipantQos** (p. 1726) and the documented default are printed. If you wish to print everything regardless, see **DDS_DomainParticipantQos_to_string_w_params** (p. 81). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1926).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_RETCODE_OK (p. 1222) if no error was encountered.

5.4.4.4 DDS_DomainParticipantQos_to_string()

```
DDS_ReturnCode_t DDS_DomainParticipantQos_to_string (
    const struct DDS_DomainParticipantQos * self,
    char * string,
    DDS_UnsignedLong * string_size )
```

Obtains a string representation of this **DDS_DomainParticipantQos** (p. 1726).

Only the differences between this **DDS_DomainParticipantQos** (p. 1726) and the documented default are printed. If you wish to print everything regardless, see **DDS_DomainParticipantQos_to_string_w_params** (p. 81). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1926).

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. 1726) is written to the buffer.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>string</i>	<< <i>out</i> >> (p. 1013) The char buffer that will be used to store the string representation of this DDS_DomainParticipantQos (p. 1726). If NULL, this function will return the required length of this buffer through the <i>string_size</i> parameter.
<i>string_size</i>	<< <i>inout</i> >> (p. 1013) 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. 1222) if no error was encountered.

See also

DDS_DomainParticipantQos_to_string_w_params (p. 81)

5.4.4.5 DDS_DomainParticipantQos_to_string_w_params()

```
DDS_ReturnCode_t DDS_DomainParticipantQos_to_string_w_params (
    const struct DDS_DomainParticipantQos * self,
    char * string,
    DDS_UnsignedLong * string_size,
    const struct DDS_DomainParticipantQos * base,
    const struct DDS_QosPrintFormat * format )
```

Obtains a string representation of this **DDS_DomainParticipantQos** (p. 1726).

Only the differences between this **DDS_DomainParticipantQos** (p. 1726) and the **DDS_DomainParticipantQos** (p. 1726) supplied as the base are printed to the string.

It is possible to supply the sentinel value **DDS_DOMAINPARTICIPANT_QOS_PRINT_ALL** (p. 160) as the base to print everything within the QoS.

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. 1726) is written to the buffer.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>string</i>	<< <i>out</i> >> (p. 1013) The char buffer that will be used to store the string representation of this DDS_DomainParticipantQos (p. 1726). If NULL, this function will return the required length of this buffer through the <code>string_size</code> parameter.
<i>string_size</i>	<< <i>inout</i> >> (p. 1013) 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.
<i>base</i>	<< <i>in</i> >> (p. 1013) The DDS_DomainParticipantQos (p. 1726) to be used as the base QoS profile. Only the differences with respect to this base profile will be included in the output string.
<i>format</i>	<< <i>in</i> >> (p. 1013) The DDS_QosPrintFormat (p. 1926) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 1222) if no error was encountered.

5.4.4.6 DDS_DomainParticipantQos_initialize()

```
DDS_ReturnCode_t DDS_DomainParticipantQos_initialize (
    struct DDS_DomainParticipantQos * self )
```

Initializer for new QoS instances.

New **DDS_DomainParticipantQos** (p. 1726) instances on heap should be initialized with this function before they are passed to any functions. This step ensures that those contained QoS policies that use dynamic memory are properly initialized. This function does not allocate memory.

Calling this function is not a substitute for calling **DDS_DomainParticipant_get_qos** (p. 152) or **DDS_DomainParticipantFactory_get_default_participant_qos** (p. 41); one of those functions should be called subsequently to setting the QoS of any new or existing entity. **DDS_DomainParticipantQos_finalize** (p. 82) should be called to free the contained QoS policies that use dynamic memory:

```
DDS_DomainParticipantQos *myQos = malloc(sizeof(struct DDS_DomainParticipantQos));

DDS_DomainParticipantQos_initialize(myQos);

DDS_DomainParticipantFactory_get_default_participant_qos(myFactory, myQos);

DDS_DomainParticipant_set_qos(myParticipant, myQos);

DDS_DomainParticipantQos_finalize(myQos);

free(myQos);
```

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DomainParticipantFactory_get_default_participant_qos (p. 41)

DDS_DomainParticipantQos_finalize (p. 82)

5.4.4.7 DDS_DomainParticipantQos_finalize()

```
DDS_ReturnCode_t DDS_DomainParticipantQos_finalize (
    struct DDS_DomainParticipantQos * self )
```

Free any dynamic memory allocated by the policies in this **DDS_DomainParticipantQos** (p. 1726).

Some QoS policies may use dynamic memory (regardless of whether the QoS itself is in dynamic memory). This function frees that memory but otherwise leaves this QoS unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

This function does not leave this object in an invalid state. It is permissible to finalize a QoS and then subsequently allocate new dynamic memory in one or more of its QoS policies.

Note that if this QoS instance is stored in heap memory, calling this function will *not* call free() on it; the user is responsible for explicitly freeing any heap-based QoS instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DomainParticipantQos_INITIALIZER (p. 75)

DDS_DomainParticipantQos_initialize (p. 81)

5.4.4.8 DDS_DomainParticipantQos_copy()

```
DDS_ReturnCode_t DDS_DomainParticipantQos_copy (
    struct DDS_DomainParticipantQos * self,
    const struct DDS_DomainParticipantQos * source )
```

Copy the contents of the given QoS into this QoS.

DDS_DomainParticipantQos (p. 1726) instances can use dynamic memory because of the sequences contained in some QoS policies. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). QoS to be copied from.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DomainParticipantQos_INITIALIZER (p. 75)

DDS_DomainParticipantQos_initialize (p. 81)

DDS_DomainParticipantQos_finalize (p. 82)

5.4.4.9 DDS_DomainParticipant_as_entity()

```
DDS_Entity * DDS_DomainParticipant_as_entity (
    DDS_DomainParticipant * domain )
```

Access a **DDS_DomainParticipant** (p. 77)'s supertype instance.

Parameters

<i>domain</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_DomainParticipant (p. 77)'s supertype **DDS_Entity** (p. 1360) instance

5.4.4.10 DDS_DomainParticipantProtocolStatus_initialize()

```
DDS_ReturnCode_t DDS_DomainParticipantProtocolStatus_initialize (
    struct DDS_DomainParticipantProtocolStatus * self )
```

Initializer for new status instances.

New **DDS_DomainParticipantProtocolStatus** (p. 1725) instance on heap should be initialized with this function before they are passed to any function. This step ensures that those contained fields that use dynamic memory are properly initialized. This function does not allocate memory.

DDS_DomainParticipantProtocolStatus_finalize (p. 85) should be called to free the contained fields that use dynamic memory:

```
DDS_DomainParticipantProtocolStatus *myStatus = malloc(sizeof(struct DDS_DomainParticipantProtocolStatus));
DDS_DomainParticipantProtocolStatus_initialize(myStatus);
DDS_DomainParticipant_get_participant_protocol_status(myParticipant, myStatus);
DDS_DomainParticipantProtocolStatus_finalize(myStatus);
free(myStatus);
```

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DomainParticipant_get_participant_protocol_status (p. 128)

DDS_DomainParticipantProtocolStatus_finalize (p. 85)

5.4.4.11 DDS_DomainParticipantProtocolStatus_copy()

```
DDS_ReturnCode_t DDS_DomainParticipantProtocolStatus_copy (
    struct DDS_DomainParticipantProtocolStatus * self,
    const struct DDS_DomainParticipantProtocolStatus * source )
```

Copy the contents of the given status into this status.

Status instances can use dynamic memory because of the sequences contained in some status. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). Status to be copied from.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DomainParticipantProtocolStatus_INITIALIZER (p. 76)

DDS_DomainParticipantProtocolStatus_initialize (p. 84)

DDS_DomainParticipantProtocolStatus_finalize (p. 85)

5.4.4.12 DDS_DomainParticipantProtocolStatus_finalize()

```
DDS_ReturnCode_t DDS_DomainParticipantProtocolStatus_finalize (
    struct DDS_DomainParticipantProtocolStatus * self )
```

Free any dynamic memory allocated by status instances.

Some status may use dynamic memory (regardless of whether the status itself is in dynamic memory). This function frees that memory but otherwise leaves this status unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

Note that if this status instance is stored in heap memory, calling this function will *not* call free() on it; the user is responsible for explicitly freeing any heap-based status instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DomainParticipantProtocolStatus_INITIALIZER (p. 76)

DDS_DomainParticipantProtocolStatus_initialize (p. 84)

5.4.4.13 DDS_DomainParticipantProtocolStatus_equals()

```
DDS_Boolean DDS_DomainParticipantProtocolStatus_equals (
    const struct DDS_DomainParticipantProtocolStatus * left,
    const struct DDS_DomainParticipantProtocolStatus * right )
```

Compares two **DDS_DomainParticipantProtocolStatus** (p. 1725) for equality.

Parameters

<i>left</i>	<< <i>in</i> >> (p. 1013) This DomainParticipantProtocolStatus. Can be NULL.
<i>right</i>	<< <i>in</i> >> (p. 1013) The other DomainParticipantProtocolStatus to be compared with this status instance. Can be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two DomainParticipantProtocolStatus have equal values, or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.4.4.14 DDS_DomainParticipant_get_default_topic_qos()

```
DDS_ReturnCode_t DDS_DomainParticipant_get_default_topic_qos (
    DDS_DomainParticipant * self,
    struct DDS_TopicQos * qos )
```

Copies the default **DDS_TopicQos** (p. 2042) values for this domain participant into the given **DDS_TopicQos** (p. 2042) instance.

The retrieved *qos* will match the set of values specified on the last successful call to **DDS_DomainParticipant_set_default_topic_qos** (p. 87), or **DDS_DomainParticipant_set_default_topic_qos_with_profile** (p. 88) or else, if the call was never made, the default values listed in **DDS_TopicQos** (p. 2042).

This function 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 **DDS_DomainParticipant_set_default_topic_qos** (p. 87)

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>in</i> >> (p. 1013) Default qos to be retrieved. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_TOPIC_QOS_DEFAULT (p. 160)

DDS_DomainParticipant_create_topic (p. 115)

5.4.4.15 DDS_DomainParticipant_set_default_topic_qos()

```
DDS_ReturnCode_t DDS_DomainParticipant_set_default_topic_qos (
    DDS_DomainParticipant * self,
    const struct DDS_TopicQos * qos )
```

Set the default **DDS_TopicQos** (p. 2042) values for this domain participant.

This default value will be used for newly created **DDS_Topic** (p. 175) if **DDS_TOPIC_QOS_DEFAULT** (p. 160) is specified as the *qos* parameter when **DDS_DomainParticipant_create_topic** (p. 115) 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. 1222)

MT Safety:

UNSAFE. It is not safe to set the default topic QoS for a DomainParticipant while another thread may be simultaneously calling **DDS_DomainParticipant_set_default_topic_qos** (p. 87), **DDS_DomainParticipant_get_default_topic_qos** (p. 86) or calling **DDS_DomainParticipant_create_topic** (p. 115) with **DDS_TOPIC_QOS_DEFAULT** (p. 160) as the *qos* parameter.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>in</i> >> (p. 1013) Default qos to be set. The special value DDS_TOPIC_QOS_DEFAULT (p. 160) may be passed as <i>qos</i> to indicate that the default QoS should be reset back to the initial values the factory would use if DDS_DomainParticipant_set_default_topic_qos (p. 87) had never been called. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222)

See also

DDS_TOPIC_QOS_DEFAULT (p. 160)

DDS_DomainParticipant_create_topic (p. 115)

5.4.4.16 DDS_DomainParticipant_set_default_topic_qos_with_profile()

```
DDS_ReturnCode_t DDS_DomainParticipant_set_default_topic_qos_with_profile (
    DDS_DomainParticipant * self,
    const char * library_name,
    const char * profile_name )
```

<<**extension**>> (p. 1012) Set the default **DDS_TopicQos** (p. 2042) values for this domain participant based on the input XML QoS profile.

This default value will be used for newly created **DDS_Topic** (p. 175) if **DDS_TOPIC_QOS_DEFAULT** (p. 160) is specified as the `qos` parameter when **DDS_DomainParticipant_create_topic** (p. 115) is called.

Precondition

The **DDS_TopicQos** (p. 2042) 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. 1222)

MT Safety:

UNSAFE. It is not safe to set the default topic QoS for a DomainParticipant while another thread may be simultaneously calling **DDS_DomainParticipant_set_default_topic_qos** (p. 87), **DDS_DomainParticipant_get_default_topic_qos** (p. 86) or calling **DDS_DomainParticipant_create_topic** (p. 115) with **DDS_TOPIC_QOS_DEFAULT** (p. 160) as the `qos` parameter.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) Library name containing the XML QoS profile. If <code>library_name</code> is null RTI Connexx will use the default library (see DDS_DomainParticipant_set_default_library (p. 101)).
<i>profile_name</i>	<< in >> (p. 1013) XML QoS Profile name. If <code>profile_name</code> is null RTI Connexx will use the default profile (see DDS_DomainParticipant_set_default_profile (p. 102)).

If the input profile cannot be found the function fails with **DDS_RETCODE_ERROR** (p. 1222).

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222)

See also

DDS_TOPIC_QOS_DEFAULT (p. 160)

DDS_DomainParticipant_create_topic_with_profile (p. 116)

5.4.4.17 DDS_DomainParticipant_get_default_publisher_qos()

```
DDS_ReturnCode_t DDS_DomainParticipant_get_default_publisher_qos (
    DDS_DomainParticipant * self,
    struct DDS_PublisherQos * qos )
```

Copy the default **DDS_PublisherQos** (p. 1918) values into the provided **DDS_PublisherQos** (p. 1918) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **DDS_DomainParticipant_set_default_publisher_qos** (p. 89), or **DDS_DomainParticipant_set_default_publisher_qos_with_profile** (p. 90), or else, if the call was never made, the default values listed in **DDS_PublisherQos** (p. 1918).

This function may potentially allocate memory depending on the sequences contained in some QoS policies.

If **DDS_PUBLISHER_QOS_DEFAULT** (p. 161) is specified as the `qos` parameter when **DDS_DomainParticipant_create_topic** (p. 115) is called, the default value of the QoS set in the factory, equivalent to the value obtained by calling **DDS_DomainParticipant_get_default_publisher_qos** (p. 89), will be used to create the **DDS_Publisher** (p. 443).

MT Safety:

UNSAFE. It is not safe to retrieve the default publisher QoS from a DomainParticipant while another thread may be simultaneously calling **DDS_DomainParticipant_set_default_publisher_qos** (p. 89)

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>inout</i> >> (p. 1013) Qos to be filled up. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_PUBLISHER_QOS_DEFAULT (p. 161)

DDS_DomainParticipant_create_publisher (p. 103)

5.4.4.18 DDS_DomainParticipant_set_default_publisher_qos()

```
DDS_ReturnCode_t DDS_DomainParticipant_set_default_publisher_qos (
    DDS_DomainParticipant * self,
    const struct DDS_PublisherQos * qos )
```

Set the default **DDS_PublisherQos** (p. 1918) values for this DomainParticipant.

This set of default values will be used for a newly created **DDS_Publisher** (p. 443) if **DDS_PUBLISHER_QOS_↵**
DEFAULT (p. 161) is specified as the `qos` parameter when **DDS_DomainParticipant_create_publisher** (p. 103) 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. 1222)

MT Safety:

UNSAFE. It is not safe to set the default publisher QoS for a DomainParticipant while another thread may be simultaneously calling **DDS_DomainParticipant_set_default_publisher_qos** (p. 89), **DDS_DomainParticipant_↵**
_get_default_publisher_qos (p. 89) or calling **DDS_DomainParticipant_create_publisher** (p. 103) with **DDS_↵**
_PUBLISHER_QOS_DEFAULT (p. 161) as the `qos` parameter.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>in</i> >> (p. 1013) Default qos to be set. The special value DDS_PUBLISHER_QOS_DEFAULT (p. 161) may be passed as <code>qos</code> to indicate that the default QoS should be reset back to the initial values the factory would use if DDS_DomainParticipant_set_default_publisher_qos (p. 89) had never been called. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222)

See also

DDS_PUBLISHER_QOS_DEFAULT (p. 161)

DDS_DomainParticipant_create_publisher (p. 103)

5.4.4.19 DDS_DomainParticipant_set_default_publisher_qos_with_profile()

```
DDS_ReturnCode_t DDS_DomainParticipant_set_default_publisher_qos_with_profile (
    DDS_DomainParticipant * self,
```

```
const char * library_name,
const char * profile_name )
```

<<**extension**>> (p. 1012) Set the default **DDS_PublisherQos** (p. 1918) values for this DomainParticipant based on the input XML QoS profile.

This set of default values will be used for a newly created **DDS_Publisher** (p. 443) if **DDS_PUBLISHER_QOS_↵**
DEFAULT (p. 161) is specified as the `qos` parameter when **DDS_DomainParticipant_create_publisher** (p. 103) is called.

Precondition

The **DDS_PublisherQos** (p. 1918) 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. 1222)

MT Safety:

UNSAFE. It is not safe to set the default publisher QoS for a DomainParticipant while another thread may be simultaneously calling **DDS_DomainParticipant_set_default_publisher_qos** (p. 89), **DDS_DomainParticipant_↵**
_get_default_publisher_qos (p. 89) or calling **DDS_DomainParticipant_create_publisher** (p. 103) with **DDS_↵**
PUBLISHER_QOS_DEFAULT (p. 161) as the `qos` parameter.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) Library name containing the XML QoS profile. If <code>library_name</code> is null RTI Connex will use the default library (see DDS_DomainParticipant_set_default_library (p. 101)).
<i>profile_name</i>	<< in >> (p. 1013) XML QoS Profile name. If <code>profile_name</code> is null RTI Connex will use the default profile (see DDS_DomainParticipant_set_default_profile (p. 102)).

If the input profile cannot be found, the function fails with **DDS_RETCODE_ERROR** (p. 1222).

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222)

See also

DDS_PUBLISHER_QOS_DEFAULT (p. 161)

DDS_DomainParticipant_create_publisher_with_profile (p. 104)

5.4.4.20 DDS_DomainParticipant_get_default_datawriter_qos()

```
DDS_ReturnCode_t DDS_DomainParticipant_get_default_datawriter_qos (
    DDS_DomainParticipant * self,
    struct DDS_DataWriterQos * qos )
```

<<**extension**>> (p. 1012) Copy the default **DDS_DataWriterQos** (p. 1666) values into the provided **DDS_DataWriterQos** (p. 1666) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **DDS_DomainParticipant_set_default_datawriter_qos** (p. 92), or **DDS_DomainParticipant_set_default_datawriter_qos_with_profile** (p. 93), or else, if the call was never made, the default values listed in **DDS_DataWriterQos** (p. 1666).

This function 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 DomainParticipant while another thread may be simultaneously calling **DDS_DomainParticipant_set_default_datawriter_qos** (p. 92).

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< inout >> (p. 1013) Qos to be filled up. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.4.4.21 DDS_DomainParticipant_set_default_datawriter_qos()

```
DDS_ReturnCode_t DDS_DomainParticipant_set_default_datawriter_qos (
    DDS_DomainParticipant * self,
    const struct DDS_DataWriterQos * qos )
```

<<**extension**>> (p. 1012) Set the default DataWriterQos values for this DomainParticipant.

This set of default values will be inherited for a newly created **DDS_Publisher** (p. 443).

Precondition

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222)

MT Safety:

UNSAFE. It is not safe to set the default DataWriter QoS for a DomainParticipant while another thread may be simultaneously calling **DDS_DomainParticipant_set_default_datawriter_qos** (p. 92) or **DDS_DomainParticipant_get_default_datawriter_qos** (p. 91) or calling **DDS_DomainParticipant_create_datawriter** (p. 108) with **DDS_DATAWRITER_QOS_DEFAULT** (p. 470) as the `qos` parameter.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>in</i> >> (p. 1013) Default qos to be set. The special value DDS_DATAWRITER_QOS_DEFAULT (p. 470) may be passed as <i>qos</i> to indicate that the default QoS should be reset back to the initial values the factory would use if DDS_DomainParticipant_set_default_datawriter_qos (p. 92) had never been called. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222)

5.4.4.22 DDS_DomainParticipant_set_default_datawriter_qos_with_profile()

```
DDS_ReturnCode_t DDS_DomainParticipant_set_default_datawriter_qos_with_profile (
    DDS_DomainParticipant * self,
    const char * library_name,
    const char * profile_name )
```

<<**extension**>> (p. 1012) Set the default **DDS_DataWriterQos** (p. 1666) values for this domain participant based on the input XML QoS profile.

This set of default values will be inherited for a newly created **DDS_Publisher** (p. 443).

Precondition

The **DDS_DataWriterQos** (p. 1666) 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. 1222)

MT Safety:

UNSAFE. It is not safe to set the default DataWriter QoS for a DomainParticipant while another thread may be simultaneously calling **DDS_DomainParticipant_set_default_datawriter_qos** (p. 92) or **DDS_DomainParticipant_get_default_datawriter_qos** (p. 91)

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< <i>in</i> >> (p. 1013) Library name containing the XML QoS profile. If <i>library_name</i> is null RTI Connext will use the default library (see DDS_DomainParticipant_set_default_library (p. 101)).
<i>profile_name</i>	<< <i>in</i> >> (p. 1013) XML QoS Profile name. If <i>profile_name</i> is null RTI Connext will use the default profile (see DDS_DomainParticipant_set_default_profile (p. 102)).

If the input profile cannot be found, the function fails with **DDS_RETCODE_ERROR** (p. 1222).

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222)

5.4.4.23 DDS_DomainParticipant_get_default_subscriber_qos()

```
DDS_ReturnCode_t DDS_DomainParticipant_get_default_subscriber_qos (
    DDS_DomainParticipant * self,
    struct DDS_SubscriberQos * qos )
```

Copy the default **DDS_SubscriberQos** (p. 2010) values into the provided **DDS_SubscriberQos** (p. 2010) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **DDS_DomainParticipant_set_default_subscriber_qos** (p. 94), or **DDS_DomainParticipant_set_default_subscriber_qos_with_profile** (p. 95), or else, if the call was never made, the default values listed in **DDS_SubscriberQos** (p. 2010).

This function may potentially allocate memory depending on the sequences contained in some QoS policies.

If **DDS_SUBSCRIBER_QOS_DEFAULT** (p. 162) is specified as the `qos` parameter when **DDS_DomainParticipant_create_subscriber** (p. 106) is called, the default value of the QoS set in the factory, equivalent to the value obtained by calling **DDS_DomainParticipant_get_default_subscriber_qos** (p. 94), will be used to create the **DDS_Subscriber** (p. 572).

MT Safety:

UNSAFE. It is not safe to retrieve the default Subscriber QoS from a DomainParticipant while another thread may be simultaneously calling **DDS_DomainParticipant_set_default_subscriber_qos** (p. 94).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>inout</i> >> (p. 1013) Qos to be filled up. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_SUBSCRIBER_QOS_DEFAULT (p. 162)

DDS_DomainParticipant_create_subscriber (p. 106)

5.4.4.24 DDS_DomainParticipant_set_default_subscriber_qos()

```
DDS_ReturnCode_t DDS_DomainParticipant_set_default_subscriber_qos (
    DDS_DomainParticipant * self,
    const struct DDS_SubscriberQos * qos )
```

Set the default **DDS_SubscriberQos** (p. 2010) values for this DomainParticipant.

This set of default values will be used for a newly created **DDS_Subscriber** (p. 572) if **DDS_SUBSCRIBER_QOS_DEFAULT** (p. 162) is specified as the `qos` parameter when **DDS_DomainParticipant_create_subscriber** (p. 106) 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. 1222)

MT Safety:

UNSAFE. It is not safe to set the default Subscriber QoS for a DomainParticipant while another thread may be simultaneously calling **DDS_DomainParticipant_set_default_subscriber_qos** (p. 94), **DDS_DomainParticipant_get_default_subscriber_qos** (p. 94) or calling **DDS_DomainParticipant_create_subscriber** (p. 106) with **DDS_SUBSCRIBER_QOS_DEFAULT** (p. 162) as the `qos` parameter.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>in</i> >> (p. 1013) Default qos to be set. The special value DDS_SUBSCRIBER_QOS_DEFAULT (p. 162) may be passed as <code>qos</code> to indicate that the default QoS should be reset back to the initial values the factory would used if DDS_DomainParticipant_set_default_subscriber_qos (p. 94) had never been called. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222)

5.4.4.25 DDS_DomainParticipant_set_default_subscriber_qos_with_profile()

```
DDS_ReturnCode_t DDS_DomainParticipant_set_default_subscriber_qos_with_profile (
    DDS_DomainParticipant * self,
    const char * library_name,
    const char * profile_name )
```

<<*extension*>> (p. 1012) Set the default **DDS_SubscriberQos** (p. 2010) values for this DomainParticipant based on the input XML QoS profile.

This set of default values will be used for a newly created **DDS_Subscriber** (p. 572) if **DDS_SUBSCRIBER_QOS_DEFAULT** (p. 162) is specified as the `qos` parameter when **DDS_DomainParticipant_create_subscriber** (p. 106) is called.

Precondition

The **DDS_SubscriberQos** (p.2010) 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. 1222)

MT Safety:

UNSAFE. It is not safe to set the default Subscriber QoS for a DomainParticipant while another thread may be simultaneously calling **DDS_DomainParticipant_set_default_subscriber_qos** (p. 94), **DDS_DomainParticipant_get_default_subscriber_qos** (p. 94) or calling **DDS_DomainParticipant_create_subscriber** (p. 106) with **DDS_SUBSCRIBER_QOS_DEFAULT** (p. 162) as the qos parameter.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< <i>in</i> >> (p. 1013) Library name containing the XML QoS profile. If library_name is null RTI Connex will use the default library (see DDS_DomainParticipant_set_default_library (p. 101)).
<i>profile_name</i>	<< <i>in</i> >> (p. 1013) XML QoS Profile name. If profile_name is null RTI Connex will use the default profile (see DDS_DomainParticipant_set_default_profile (p. 102)).

If the input profile cannot be found, the function fails with **DDS_RETCODE_ERROR** (p. 1222).

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222)

See also

DDS_SUBSCRIBER_QOS_DEFAULT (p. 162)

DDS_DomainParticipant_create_subscriber_with_profile (p. 107)

5.4.4.26 DDS_DomainParticipant_get_default_datareader_qos()

```
DDS_ReturnCode_t DDS_DomainParticipant_get_default_datareader_qos (
    DDS_DomainParticipant * self,
    struct DDS_DataReaderQos * qos )
```

<<*extension*>> (p. 1012) Copy the default **DDS_DataReaderQos** (p. 1611) values into the provided **DDS_DataReaderQos** (p. 1611) instance.

The retrieved qos will match the set of values specified on the last successful call to **DDS_DomainParticipant_set_default_datareader_qos** (p. 97), or **DDS_DomainParticipant_set_default_datareader_qos_with_profile** (p. 97), or else, if the call was never made, the default values listed in **DDS_DataReaderQos** (p. 1611).

This function 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 **DDS_DomainParticipant_set_default_datareader_qos** (p. 97).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>inout</i> >> (p. 1013) Qos to be filled up. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.4.4.27 DDS_DomainParticipant_set_default_datareader_qos()

```
DDS_ReturnCode_t DDS_DomainParticipant_set_default_datareader_qos (
    DDS_DomainParticipant * self,
    const struct DDS_DataReaderQos * qos )
```

<<*extension*>> (p. 1012) Set the default **DDS_DataReaderQos** (p. 1611) values for this domain participant.

This set of default values will be inherited for a newly created **DDS_Subscriber** (p. 572).

Precondition

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **DDS_↵
RETCODE_INCONSISTENT_POLICY** (p. 1222)

MT Safety:

UNSAFE. It is not safe to set the default DataReader QoS for a DomainParticipant while another thread may be simultaneously calling **DDS_DomainParticipant_set_default_datareader_qos** (p. 97) or **DDS_Domain↵
Participant_get_default_datareader_qos** (p. 96).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>in</i> >> (p. 1013) Default qos to be set. The special value DDS_DATAREADER_QOS_DEFAULT (p. 600) may be passed as <i>qos</i> to indicate that the default QoS should be reset back to the initial values the factory would use if DDS_DomainParticipant_set_default_datareader_qos (p. 97) had never been called. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222)

5.4.4.28 DDS_DomainParticipant_set_default_datareader_qos_with_profile()

```
DDS_ReturnCode_t DDS_DomainParticipant_set_default_datareader_qos_with_profile (
    DDS_DomainParticipant * self,
    const char * library_name,
    const char * profile_name )
```

<<**extension**>> (p. 1012) Set the default **DDS_DataReaderQos** (p. 1611) values for this DomainParticipant based on the input XML QoS profile.

This set of default values will be inherited for a newly created **DDS_Subscriber** (p. 572).

Precondition

The **DDS_DataReaderQos** (p. 1611) 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. 1222)

MT Safety:

UNSAFE. It is not safe to set the default DataReader QoS for a DomainParticipant while another thread may be simultaneously calling **DDS_DomainParticipant_set_default_datareader_qos** (p. 97) or **DDS_DomainParticipant_get_default_datareader_qos** (p. 96).

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) Library name containing the XML QoS profile. If library_name is null RTI Connexx will use the default library (see DDS_DomainParticipant_set_default_library (p. 101)).
<i>profile_name</i>	<< in >> (p. 1013) XML QoS Profile name. If profile_name is null RTI Connexx will use the default profile (see DDS_DomainParticipant_set_default_profile (p. 102)).

If the input profile cannot be found, the function fails with **DDS_RETCODE_ERROR** (p. 1222).

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222)

5.4.4.29 DDS_DomainParticipant_get_default_flowcontroller_property()

```
DDS_ReturnCode_t DDS_DomainParticipant_get_default_flowcontroller_property (
    DDS_DomainParticipant * self,
    struct DDS_FlowControllerProperty_t * prop )
```

<<**extension**>> (p. 1012) Copies the default **DDS_FlowControllerProperty_t** (p. 1799) values for this domain participant into the given **DDS_FlowControllerProperty_t** (p. 1799) instance.

The retrieved *property* will match the set of values specified on the last successful call to **DDS_DomainParticipant_set_default_flowcontroller_property** (p. 99), or else, if the call was never made, the default values listed in **DDS_FlowControllerProperty_t** (p. 1799).

MT Safety:

UNSAFE. It is not safe to retrieve the default flow controller properties from a DomainParticipant while another thread may be simultaneously calling **DDS_DomainParticipant_set_default_flowcontroller_property** (p. 99)

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>prop</i>	<< <i>in</i> >> (p. 1013) Default property to be retrieved. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT (p. 162)

DDS_DomainParticipant_create_flowcontroller (p. 124)

5.4.4.30 DDS_DomainParticipant_set_default_flowcontroller_property()

```
DDS_ReturnCode_t DDS_DomainParticipant_set_default_flowcontroller_property (
    DDS_DomainParticipant * self,
    const struct DDS_FlowControllerProperty_t * prop )
```

<<*extension*>> (p. 1012) Set the default **DDS_FlowControllerProperty_t** (p. 1799) values for this domain participant.

This default value will be used for newly created **DDS_FlowController** (p. 559) if **DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT** (p. 162) is specified as the `property` parameter when **DDS_DomainParticipant_create_flowcontroller** (p. 124) 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. 1222)

MT Safety:

UNSAFE. It is not safe to set the default flow controller properties for a DomainParticipant while another thread may be simultaneously calling **DDS_DomainParticipant_set_default_flowcontroller_property** (p. 99) , **DDS_DomainParticipant_get_default_flowcontroller_property** (p. 98) or calling **DDS_DomainParticipant_create_flowcontroller** (p. 124) with **DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT** (p. 162) as the `qos` parameter.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>prop</i>	<< <i>in</i> >> (p. 1013) Default property to be set. The special value DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT (p. 162) may be passed as <code>property</code> to indicate that the default property should be reset to the default values the factory would use if DDS_DomainParticipant_set_default_flowcontroller_property (p. 99) had never been called. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222)

See also

DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT (p. 162)

DDS_DomainParticipant_create_flowcontroller (p. 124)

5.4.4.31 DDS_DomainParticipant_get_default_library()

```
const char * DDS_DomainParticipant_get_default_library (
    DDS_DomainParticipant * self )
```

<<*extension*>> (p. 1012) Gets the default XML library associated with a **DDS_DomainParticipant** (p. 77).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

The default library or null if the default library was not set.

See also

DDS_DomainParticipant_set_default_library (p. 101)

5.4.4.32 DDS_DomainParticipant_get_default_profile()

```
const char * DDS_DomainParticipant_get_default_profile (
    DDS_DomainParticipant * self )
```

<<*extension*>> (p. 1012) Gets the default XML profile associated with a **DDS_DomainParticipant** (p. 77).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

The default profile or null if the default profile was not set.

See also

DDS_DomainParticipant_set_default_profile (p. 102)

5.4.4.33 DDS_DomainParticipant_get_default_profile_library()

```
const char * DDS_DomainParticipant_get_default_profile_library (
    DDS_DomainParticipant * self )
```

<<**extension**>> (p. 1012) Gets the library where the default XML QoS profile is contained for a **DDS_DomainParticipant** (p. 77).

The default profile library is automatically set when **DDS_DomainParticipant_set_default_profile** (p. 102) is called.

This library can be different than the **DDS_DomainParticipant** (p. 77) default library (see **DDS_DomainParticipant_get_default_library** (p. 100)).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

The default profile library or null if the default profile was not set.

See also

DDS_DomainParticipant_set_default_profile (p. 102)

5.4.4.34 DDS_DomainParticipant_set_default_library()

```
DDS_ReturnCode_t DDS_DomainParticipant_set_default_library (
    DDS_DomainParticipant * self,
    const char * library_name )
```

<<**extension**>> (p. 1012) Sets the default XML library for a **DDS_DomainParticipant** (p. 77).

This function 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 **DDS_DomainParticipant** (p. 77) inherits the default from the **DDS_DomainParticipantFactory** (p. 32) (see **DDS_DomainParticipantFactory_set_default_library** (p. 48)).

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) Library name. If <code>library_name</code> is null any previous default is unset.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DomainParticipant_get_default_library (p. 100)

5.4.4.35 DDS_DomainParticipant_set_default_profile()

```
DDS_ReturnCode_t DDS_DomainParticipant_set_default_profile (
    DDS_DomainParticipant * self,
    const char * library_name,
    const char * profile_name )
```

<<**extension**>> (p. 1012) Sets the default XML profile for a **DDS_DomainParticipant** (p. 77).

This function 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 **DDS_DomainParticipant** (p. 77) function 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 **DDS_DomainParticipant** (p. 77) inherits the default from the **DDS_DomainParticipantFactory** (p. 32) (see **DDS_DomainParticipantFactory_set_default_profile** (p. 49)).

This function does not set the default QoS for entities created by the **DDS_DomainParticipant** (p. 77); for this functionality, use the functions `set_default_<entity>_qos_with_profile` (you may pass in NULL after having called `set_default_<entity>_profile()`).

This function does not set the default QoS for newly created DomainParticipants; for this functionality, use **DDS_DomainParticipantFactory_set_default_participant_qos_with_profile** (p. 40).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< <i>in</i> >> (p. 1013) The library name containing the profile.
<i>profile_name</i>	<< <i>in</i> >> (p. 1013) The profile name. If profile_name is null any previous default is unset.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DomainParticipant_get_default_profile (p. 100)

DDS_DomainParticipant_get_default_profile_library (p. 101)

5.4.4.36 DDS_DomainParticipant_create_publisher()

```
DDS_Publisher * DDS_DomainParticipant_create_publisher (
    DDS_DomainParticipant * self,
    const struct DDS_PublisherQos * qos,
    const struct DDS_PublisherListener * listener,
    DDS_StatusMask mask )
```

Creates a **DDS_Publisher** (p. 443) with the desired QoS policies and attaches to it the specified **DDS_Publisher**↔
Listener (p. 1917).

Precondition

The specified QoS policies must be consistent, or the operation will fail and no **DDS_Publisher** (p. 443) will be created.

if *listener* is specified, none of the listener callback functions can be NULL.

MT Safety:

UNSAFE. If **DDS_PUBLISHER_QOS_DEFAULT** (p. 161) is used for *qos*, it is not safe to create the publisher while another thread may be simultaneously calling **DDS_DomainParticipant_set_default_publisher_qos** (p. 89).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>in</i> >> (p. 1013) QoS to be used for creating the new DDS_Publisher (p. 443). The special value DDS_PUBLISHER_QOS_DEFAULT (p. 161) can be used to indicate that the DDS_Publisher (p. 443) should be created with the default DDS_PublisherQos (p. 1918) set in the DDS_DomainParticipant (p. 77). Cannot be NULL.
<i>listener</i>	<< <i>in</i> >> (p. 1013). Listener to be attached to the newly created DDS_Publisher (p. 443).
<i>mask</i>	<< <i>in</i> >> (p. 1013). Changes of communication status to be invoked on the listener. See DDS_StatusMask (p. 1226).

Returns

newly created publisher object or NULL on failure.

See also

Specifying QoS on entities (p. 1244) for information on setting QoS before entity creation

DDS_PublisherQos (p. 1918) for rules on consistency among QoS

DDS_PUBLISHER_QOS_DEFAULT (p. 161)

DDS_DomainParticipant_create_publisher_with_profile (p. 104)

DDS_DomainParticipant_get_default_publisher_qos (p. 89)

DDS_Publisher_set_listener (p. 466)

Examples

HelloWorld_publisher.c.

5.4.4.37 DDS_DomainParticipant_create_publisher_with_profile()

```
DDS_Publisher * DDS_DomainParticipant_create_publisher_with_profile (
    DDS_DomainParticipant * self,
    const char * library_name,
    const char * profile_name,
    const struct DDS_PublisherListener * listener,
    DDS_StatusMask mask )
```

<<*extension*>> (p. 1012) Creates a new **DDS_Publisher** (p. 443) object using the **DDS_PublisherQos** (p. 1918) associated with the input XML QoS profile.

Precondition

The **DDS_PublisherQos** (p. 1918) in the input profile must be consistent, or the operation will fail and no **DDS_Publisher** (p. 443) will be created.

if *listener* is specified, none of the listener callback functions can be NULL.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< <i>in</i> >> (p. 1013) Library name containing the XML QoS profile. If <i>library_name</i> is null RTI Connex will use the default library (see DDS_DomainParticipant_set_default_library (p. 101)).
<i>profile_name</i>	<< <i>in</i> >> (p. 1013) XML QoS Profile name. If <i>profile_name</i> is null RTI Connex will use the default profile (see DDS_DomainParticipant_set_default_profile (p. 102)).
<i>listener</i>	<< <i>in</i> >> (p. 1013). Listener to be attached to the newly created DDS_Publisher (p. 443).
<i>mask</i>	<< <i>in</i> >> (p. 1013). Changes of communication status to be invoked on the listener. See DDS_StatusMask (p. 1226).

Returns

newly created publisher object or NULL on failure.

See also

Specifying QoS on entities (p. 1244) for information on setting QoS before entity creation

DDS_PublisherQos (p. 1918) for rules on consistency among QoS

DDS_DomainParticipant_create_publisher (p. 103)

DDS_DomainParticipant_get_default_publisher_qos (p. 89)

DDS_Publisher_set_listener (p. 466)

5.4.4.38 DDS_DomainParticipant_delete_publisher()

```
DDS_ReturnCode_t DDS_DomainParticipant_delete_publisher (
    DDS_DomainParticipant * self,
    DDS_Publisher * p )
```

Deletes an existing **DDS_Publisher** (p. 443).

Precondition

The **DDS_Publisher** (p. 443) must not have any attached **DDS_DataWriter** (p. 485) objects. If there are existing **DDS_DataWriter** (p. 485) objects, it will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

DDS_Publisher (p. 443) must have been created by this **DDS_DomainParticipant** (p. 77), or else it will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

Postcondition

Listener installed on the **DDS_Publisher** (p. 443) will not be called after this function 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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>p</i>	<< <i>in</i> >> (p. 1013) DDS_Publisher (p. 443) to be deleted.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

5.4.4.39 DDS_DomainParticipant_create_subscriber()

```
DDS_Subscriber * DDS_DomainParticipant_create_subscriber (
    DDS_DomainParticipant * self,
    const struct DDS_SubscriberQos * qos,
    const struct DDS_SubscriberListener * listener,
    DDS_StatusMask mask )
```

Creates a **DDS_Subscriber** (p. 572) with the desired QoS policies and attaches to it the specified **DDS_Subscriber**↵
Listener (p. 2009).

Precondition

The specified QoS policies must be consistent, or the operation will fail and no **DDS_Subscriber** (p. 572) will be created.

if *listener* is specified, none of the listener callback functions can be NULL.

MT Safety:

UNSAFE. If **DDS_SUBSCRIBER_QOS_DEFAULT** (p. 162) is used for *qos*, it is not safe to create the subscriber while another thread may be simultaneously calling **DDS_DomainParticipant_set_default_subscriber**↵
_qos (p. 94).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>in</i> >> (p. 1013) QoS to be used for creating the new DDS_Subscriber (p. 572). The special value DDS_SUBSCRIBER_QOS_DEFAULT (p. 162) can be used to indicate that the DDS_Subscriber (p. 572) should be created with the default DDS_SubscriberQos (p. 2010) set in the DDS_DomainParticipant (p. 77). Cannot be NULL.
<i>listener</i>	<< <i>in</i> >> (p. 1013). Listener to be attached to the newly created DDS_Subscriber (p. 572).
<i>mask</i>	<< <i>in</i> >> (p. 1013). Changes of communication status to be invoked on the listener. See DDS_StatusMask (p. 1226).

Returns

newly created subscriber object or NULL on failure.

See also

Specifying QoS on entities (p. 1244) for information on setting QoS before entity creation

DDS_SubscriberQos (p. 2010) for rules on consistency among QoS

DDS_SUBSCRIBER_QOS_DEFAULT (p. 162)

DDS_DomainParticipant_create_subscriber_with_profile (p. 107)

DDS_DomainParticipant_get_default_subscriber_qos (p. 94)

DDS_Subscriber_set_listener (p. 597)

Examples

HelloWorld_subscriber.c.

5.4.4.40 DDS_DomainParticipant_create_subscriber_with_profile()

```
DDS_Subscriber * DDS_DomainParticipant_create_subscriber_with_profile (
    DDS_DomainParticipant * self,
    const char * library_name,
    const char * profile_name,
    const struct DDS_SubscriberListener * listener,
    DDS_StatusMask mask )
```

<<**extension**>> (p. 1012) Creates a new **DDS_Subscriber** (p. 572) object using the **DDS_PublisherQos** (p. 1918) associated with the input XML QoS profile.

Precondition

The **DDS_SubscriberQos** (p. 2010) in the input profile must be consistent, or the operation will fail and no **DDS_Subscriber** (p. 572) will be created.

if *listener* is specified, none of the listener callback functions can be NULL.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) Library name containing the XML QoS profile. If <i>library_name</i> is null RTI Connext will use the default library (see DDS_DomainParticipant_set_default_library (p. 101)).
<i>profile_name</i>	<< in >> (p. 1013) XML QoS Profile name. If <i>profile_name</i> is null RTI Connext will use the default profile (see DDS_DomainParticipant_set_default_profile (p. 102)).
<i>listener</i>	<< in >> (p. 1013). Listener to be attached to the newly created DDS_Subscriber (p. 572).
<i>mask</i>	<< in >> (p. 1013). Changes of communication status to be invoked on the listener. See DDS_StatusMask (p. 1226).

Returns

newly created subscriber object or NULL on failure.

See also

Specifying QoS on entities (p. 1244) for information on setting QoS before entity creation

DDS_SubscriberQos (p. 2010) for rules on consistency among QoS

DDS_DomainParticipant_create_subscriber (p. 106)

DDS_DomainParticipant_get_default_subscriber_qos (p. 94)

DDS_Subscriber_set_listener (p. 597)

5.4.4.41 DDS_DomainParticipant_delete_subscriber()

```
DDS_ReturnCode_t DDS_DomainParticipant_delete_subscriber (
    DDS_DomainParticipant * self,
    DDS_Subscriber * s )
```

Deletes an existing **DDS_Subscriber** (p. 572).

Precondition

The **DDS_Subscriber** (p. 572) must not have any attached **DDS_DataReader** (p. 615) objects. If there are existing **DDS_DataReader** (p. 615) objects, it will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222)

The **DDS_Subscriber** (p. 572) must have been created by this **DDS_DomainParticipant** (p. 77), or else it will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

Postcondition

A Listener installed on the **DDS_Subscriber** (p. 572) will not be called after this function 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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>s</i>	<< <i>in</i> >> (p. 1013) DDS_Subscriber (p. 572) to be deleted.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

5.4.4.42 DDS_DomainParticipant_create_datawriter()

```

DDS_DataWriter * DDS_DomainParticipant_create_datawriter (
    DDS_DomainParticipant * self,
    DDS_Topic * topic,
    const struct DDS_DataWriterQos * qos,
    const struct DDS_DataWriterListener * listener,
    DDS_StatusMask mask )

```

<<extension>> (p. 1012) Creates a **DDS_DataWriter** (p. 485) that will be attached and belong to the implicit **DDS_Publisher** (p. 443).

Precondition

The given **DDS_Topic** (p. 175) must have been created from the same DomainParticipant as the implicit Publisher. If it was created from a different DomainParticipant, this function will fail.

The **DDS_DataWriter** (p. 485) created using this function will be associated with the implicit Publisher. This Publisher is automatically created (if it does not exist) using **DDS_PUBLISHER_QOS_DEFAULT** (p. 161) when the following functions are called: **DDS_DomainParticipant_create_datawriter** (p. 108), **DDS_DomainParticipant_create_datawriter_with_profile** (p. 110), or **DDS_DomainParticipant_get_implicit_publisher** (p. 129).

MT Safety:

UNSAFE. If **DDS_DATAWRITER_QOS_DEFAULT** (p. 470) is used for the `qos` parameter, it is not safe to create the DataWriter while another thread may be simultaneously calling **DDS_DomainParticipant_set_default_datawriter_qos** (p. 92).

Parameters

<i>self</i>	<<in>> (p. 1013) Cannot be NULL.
<i>topic</i>	<<in>> (p. 1013) The DDS_Topic (p. 175) that the DDS_DataWriter (p. 485) will be associated with. Cannot be NULL.
<i>qos</i>	<<in>> (p. 1013) QoS to be used for creating the new DDS_DataWriter (p. 485). The special value DDS_DATAWRITER_QOS_DEFAULT (p. 470) can be used to indicate that the DDS_DataWriter (p. 485) should be created with the default DDS_DataWriterQos (p. 1666) set in the implicit DDS_Publisher (p. 443). The special value DDS_DATAWRITER_QOS_USE_TOPIC_QOS (p. 470) can be used to indicate that the DDS_DataWriter (p. 485) should be created with the combination of the default DDS_DataWriterQos (p. 1666) set on the DDS_Publisher (p. 443) and the DDS_TopicQos (p. 2042) of the DDS_Topic (p. 175). Cannot be NULL.
<i>listener</i>	<<in>> (p. 1013) The listener of the DDS_DataWriter (p. 485).
<i>mask</i>	<<in>> (p. 1013). Changes of communication status to be invoked on the listener. See DDS_StatusMask (p. 1226).

Returns

A **DDS_DataWriter** (p. 485) of a derived class specific to the data type associated with the **DDS_Topic** (p. 175) or NULL if an error occurred.

See also

FooDataWriter (p. 2113)
Specifying QoS on entities (p. 1244) for information on setting QoS before entity creation
DDS_DataWriterQos (p. 1666) for rules on consistency among QoS
DDS_DATAWRITER_QOS_DEFAULT (p. 470)
DDS_DATAWRITER_QOS_USE_TOPIC_QOS (p. 470)
DDS_DomainParticipant_create_datawriter_with_profile (p. 110)
DDS_DomainParticipant_get_default_datawriter_qos (p. 91)
DDS_DomainParticipant_get_implicit_publisher (p. 129)
DDS_Topic_set_qos (p. 197)
DDS_DataWriter_set_listener (p. 553)

5.4.4.43 DDS_DomainParticipant_create_datawriter_with_profile()

```

DDS_DataWriter * DDS_DomainParticipant_create_datawriter_with_profile (
    DDS_DomainParticipant * self,
    DDS_Topic * topic,
    const char * library_name,
    const char * profile_name,
    const struct DDS_DataWriterListener * listener,
    DDS_StatusMask mask )
  
```

<<**extension**>> (p. 1012) Creates a **DDS_DataWriter** (p. 485) using a XML QoS profile that will be attached and belong to the implicit **DDS_Publisher** (p. 443).

Precondition

The given **DDS_Topic** (p. 175) must have been created from the same DomainParticipant as the implicit Publisher. If it was created from a different DomainParticipant, this function will return NULL.

The **DDS_DataWriter** (p. 485) created using this function will be associated with the implicit Publisher. This Publisher is automatically created (if it does not exist) using **DDS_PUBLISHER_QOS_DEFAULT** (p. 161) when the following functions are called: **DDS_DomainParticipant_create_datawriter** (p. 108), **DDS_DomainParticipant_create_datawriter_with_profile** (p. 110), or **DDS_DomainParticipant_get_implicit_publisher** (p. 129)

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>topic</i>	<< in >> (p. 1013) The DDS_Topic (p. 175) that the DDS_DataWriter (p. 485) will be associated with. Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) Library name containing the XML QoS profile. If library_name is null RTI Connexx will use the default library (see DDS_DomainParticipant_set_default_library (p. 101)).
<i>profile_name</i>	<< in >> (p. 1013) XML QoS Profile name. If profile_name is null RTI Connexx will use the default profile (see DDS_DomainParticipant_set_default_profile (p. 102)).
<i>listener</i>	<< in >> (p. 1013) The listener of the DDS_DataWriter (p. 485).
<i>mask</i>	<< in >> (p. 1013). Changes of communication status to be invoked on the listener. See DDS_StatusMask (p. 1226).

Returns

A **DDS_DataWriter** (p. 485) of a derived class specific to the data type associated with the **DDS_Topic** (p. 175) or NULL if an error occurred.

See also

FooDataWriter (p. 2113)

Specifying QoS on entities (p. 1244) for information on setting QoS before entity creation

DDS_DataWriterQos (p. 1666) for rules on consistency among QoS

DDS_DomainParticipant_create_datawriter (p. 108)

DDS_DomainParticipant_get_default_datawriter_qos (p. 91)

DDS_DomainParticipant_get_implicit_publisher (p. 129)

DDS_Topic_set_qos (p. 197)

DDS_DataWriter_set_listener (p. 553)

5.4.4.44 **DDS_DomainParticipant_delete_datawriter()**

```
DDS_ReturnCode_t DDS_DomainParticipant_delete_datawriter (
    DDS_DomainParticipant * self,
    DDS_DataWriter * a_datawriter )
```

<<**extension**>> (p. 1012) Deletes a **DDS_DataWriter** (p. 485) that belongs to the implicit **DDS_Publisher** (p. 443).

The deletion of the **DDS_DataWriter** (p. 485) will automatically unregister all instances.

Precondition

If the **DDS_DataWriter** (p. 485) does not belong to the implicit **DDS_Publisher** (p. 443), the operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

Postcondition

Listener installed on the **DDS_DataWriter** (p. 485) will not be called after this function 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

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>a_datawriter</i>	<< in >> (p. 1013) The DDS_DataWriter (p. 485) to be deleted.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

See also

DDS_DomainParticipant_get_implicit_publisher (p. 129)

5.4.4.45 DDS_DomainParticipant_create_datareader()

```
DDS_DataReader * DDS_DomainParticipant_create_datareader (
    DDS_DomainParticipant * self,
    DDS_TopicDescription * topic,
    const struct DDS_DataReaderQos * qos,
    const struct DDS_DataReaderListener * listener,
    DDS_StatusMask mask )
```

<<*extension*>> (p. 1012) Creates a **DDS_DataReader** (p. 615) that will be attached and belong to the implicit **DDS_←_Subscriber** (p. 572).

Precondition

The given **DDS_TopicDescription** (p. 175) must have been created from the same DomainParticipant as the implicit Subscriber. If it was created from a different DomainParticipant, this function will return NULL.

The **DDS_DataReader** (p. 615) created using this function will be associated with the implicit Subscriber. This Subscriber is automatically created (if it does not exist) using **DDS_SUBSCRIBER_QOS_DEFAULT** (p. 162) when the following functions are called: **DDS_DomainParticipant_create_datareader** (p. 112), **DDS_DomainParticipant_←create_datareader_with_profile** (p. 113), or **DDS_DomainParticipant_get_implicit_subscriber** (p. 130).

MT Safety:

UNSAFE. If **DDS_DATAREADER_QOS_DEFAULT** (p. 600) is used for the `qos` parameter, it is not safe to create the datareader while another thread may be simultaneously calling **DDS_DomainParticipant_set_default_←datareader_qos** (p. 97).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>topic</i>	<< <i>in</i> >> (p. 1013) The DDS_TopicDescription (p. 175) that the DDS_DataReader (p. 615) will be associated with. Cannot be NULL.
<i>qos</i>	<< <i>in</i> >> (p. 1013) The qos of the DDS_DataReader (p. 615). The special value DDS_DATAREADER_QOS_DEFAULT (p. 600) can be used to indicate that the DDS_DataReader (p. 615) should be created with the default DDS_DataReaderQos (p. 1611) set in the implicit DDS_Subscriber (p. 572). If DDS_TopicDescription (p. 175) is of type DDS_Topic (p. 175) or DDS_ContentFilteredTopic (p. 177), the special value DDS_DATAREADER_QOS_USE_TOPIC_QOS (p. 601) can be used to indicate that the DDS_DataReader (p. 615) should be created with the combination of the default DDS_DataReaderQos (p. 1611) set on the implicit DDS_Subscriber (p. 572) and the DDS_TopicQos (p. 2042) (in the case of a DDS_ContentFilteredTopic (p. 177), the DDS_TopicQos (p. 2042) of the related DDS_Topic (p. 175)). if DDS_DATAREADER_QOS_USE_TOPIC_QOS (p. 601) is used, <code>topic</code> cannot be a DDS_MultiTopic (p. 185). Cannot be NULL.
<i>listener</i>	<< <i>in</i> >> (p. 1013) The listener of the DDS_DataReader (p. 615).

Returns

A **DDS_DataReader** (p. 615) of a derived class specific to the data-type associated with the **DDS_Topic** (p. 175) or NULL if an error occurred.

See also

FooDataReader (p. 2112)

Specifying QoS on entities (p. 1244) for information on setting QoS before entity creation

DDS_DataReaderQos (p. 1611) for rules on consistency among QoS

DDS_DomainParticipant_create_datareader_with_profile (p. 113)

DDS_DomainParticipant_get_default_datareader_qos (p. 96)

DDS_DomainParticipant_get_implicit_subscriber (p. 130)

DDS_Topic_set_qos (p. 197)

DDS_DataReader_set_listener (p. 690)

5.4.4.46 DDS_DomainParticipant_create_datareader_with_profile()

```
DDS_DataReader * DDS_DomainParticipant_create_datareader_with_profile (
    DDS_DomainParticipant * self,
    DDS_TopicDescription * topic,
    const char * library_name,
    const char * profile_name,
    const struct DDS_DataReaderListener * listener,
    DDS_StatusMask mask )
```

<<**extension**>> (p. 1012) Creates a **DDS_DataReader** (p. 615) using a XML QoS profile that will be attached and belong to the implicit **DDS_Subscriber** (p. 572).

Precondition

The given **DDS_TopicDescription** (p. 175) must have been created from the same DomainParticipant as the implicit subscriber. If it was created from a different DomainParticipant, this function will return NULL.

The **DDS_DataReader** (p. 615) created using this function will be associated with the implicit Subscriber. This Subscriber is automatically created (if it does not exist) using **DDS_SUBSCRIBER_QOS_DEFAULT** (p. 162) when the following functions are called: **DDS_DomainParticipant_create_datareader** (p. 112), **DDS_DomainParticipant_create_datareader_with_profile** (p. 113), or **DDS_DomainParticipant_get_implicit_subscriber** (p. 130)

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>topic</i>	<< in >> (p. 1013) The DDS_TopicDescription (p. 175) that the DDS_DataReader (p. 615) will be associated with. Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) Library name containing the XML QoS profile. If library_name is null RTI Connex will use the default library (see DDS_DomainParticipant_set_default_library (p. 101)).
<i>profile_name</i>	<< in >> (p. 1013) XML QoS Profile name. If profile_name is null RTI Connex will use the default
Generated by Doxygen	profile (see DDS_DomainParticipant_set_default_profile (p. 102)).
<i>listener</i>	<< in >> (p. 1013) The listener of the DDS_DataReader (p. 615).
<i>mask</i>	<< in >> (p. 1013). Changes of communication status to be invoked on the listener. See DDS_StatusMask (p. 1226).

Returns

A **DDS_DataReader** (p. 615) of a derived class specific to the data-type associated with the **DDS_Topic** (p. 175) or NULL if an error occurred.

See also

FooDataReader (p. 2112)

Specifying QoS on entities (p. 1244) for information on setting QoS before entity creation

DDS_DataReaderQos (p. 1611) for rules on consistency among QoS

DDS_DomainParticipant_create_datareader (p. 112)

DDS_DomainParticipant_get_default_datareader_qos (p. 96)

DDS_DomainParticipant_get_implicit_subscriber (p. 130)

DDS_Topic_set_qos (p. 197)

DDS_DataReader_set_listener (p. 690)

5.4.4.47 DDS_DomainParticipant_delete_datareader()

```
DDS_ReturnCode_t DDS_DomainParticipant_delete_datareader (
    DDS_DomainParticipant * self,
    DDS_DataReader * a_datareader )
```

<<**extension**>> (p. 1012) Deletes a **DDS_DataReader** (p. 615) that belongs to the implicit **DDS_Subscriber** (p. 572).

Precondition

If the **DDS_DataReader** (p. 615) does not belong to the implicit **DDS_Subscriber** (p. 572), or if there are any existing **DDS_ReadCondition** (p. 694) or **DDS_QueryCondition** (p. 699) objects that are attached to the **DDS_DataReader** (p. 615), 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. 1222).

Postcondition

Listener installed on the **DDS_DataReader** (p. 615) will not be called after this function 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

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>a_datareader</i>	<< in >> (p. 1013) The DDS_DataReader (p. 615) to be deleted.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

See also

DDS_DomainParticipant_get_implicit_subscriber (p. 130)

5.4.4.48 DDS_DomainParticipant_create_topic()

```
DDS_Topic * DDS_DomainParticipant_create_topic (
    DDS_DomainParticipant * self,
    const char * topic_name,
    const char * type_name,
    const struct DDS_TopicQos * qos,
    const struct DDS_TopicListener * listener,
    DDS_StatusMask mask )
```

Creates a **DDS_Topic** (p. 175) with the desired QoS policies and attaches to it the specified **DDS_TopicListener** (p. 2041).

Precondition

The application is not allowed to create two **DDS_Topic** (p. 175) objects with the same `topic_name` attached to the same **DDS_DomainParticipant** (p. 77). If the application attempts this, this function will fail and return a NULL topic.

The specified QoS policies must be consistent, or the operation will fail and no **DDS_Topic** (p. 175) will be created.

Prior to creating a **DDS_Topic** (p. 175), the type must have been registered with RTI Connex. This is done using the **FooTypeSupport_register_type** (p. 221) operation on a derived class of the **DDS_TypeSupport** (p. 214) interface.

MT Safety:

UNSAFE. It is not safe to create a topic while another thread is trying to lookup that topic description with **DDS_DomainParticipant_lookup_topicdescription** (p. 127).

MT Safety:

UNSAFE. If **DDS_TOPIC_QOS_DEFAULT** (p. 160) is used for `qos`, it is not safe to create the topic while another thread may be simultaneously calling **DDS_DomainParticipant_set_default_topic_qos** (p. 87).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>topic_name</i>	<< <i>in</i> >> (p. 1013) Name for the new topic, must not exceed 255 characters. Cannot be NULL.
<i>type_name</i>	<< <i>in</i> >> (p. 1013) The type to which the new DDS_Topic (p. 175) will be bound. Cannot be NULL.
<i>qos</i>	<< <i>in</i> >> (p. 1013) QoS to be used for creating the new DDS_Topic (p. 175). The special value DDS_TOPIC_QOS_DEFAULT (p. 160) can be used to indicate that the DDS_Topic (p. 175) should be created with the default DDS_TopicQos (p. 2042) set in the DDS_DomainParticipant (p. 77). Cannot be NULL.
<i>listener</i>	<< <i>in</i> >> (p. 1013). Listener to be attached to the newly created DDS_Topic (p. 175).

Returns

newly created topic, or NULL on failure

See also

Specifying QoS on entities (p. 1244) for information on setting QoS before entity creation

DDS_TopicQos (p. 2042) for rules on consistency among QoS

DDS_TOPIC_QOS_DEFAULT (p. 160)

DDS_DomainParticipant_create_topic_with_profile (p. 116)

DDS_DomainParticipant_get_default_topic_qos (p. 86)

DDS_Topic_set_listener (p. 199)

Examples

HelloWorld_publisher.c, and **HelloWorld_subscriber.c**.

5.4.4.49 DDS_DomainParticipant_create_topic_with_profile()

```
DDS_Topic * DDS_DomainParticipant_create_topic_with_profile (
    DDS_DomainParticipant * self,
    const char * topic_name,
    const char * type_name,
    const char * library_name,
    const char * profile_name,
    const struct DDS_TopicListener * listener,
    DDS_StatusMask mask )
```

<<**extension**>> (p. 1012) Creates a new **DDS_Topic** (p. 175) object using the **DDS_PublisherQos** (p. 1918) associated with the input XML QoS profile.

Precondition

The application is not allowed to create two **DDS_TopicDescription** (p. 175) objects with the same `topic_name` attached to the same **DDS_DomainParticipant** (p. 77). If the application attempts this, this function will fail and return a NULL topic.

The **DDS_TopicQos** (p. 2042) in the input profile must be consistent, or the operation will fail and no **DDS_Topic** (p. 175) will be created.

Prior to creating a **DDS_Topic** (p. 175), the type must have been registered with RTI Connex. This is done using the **FooTypeSupport_register_type** (p. 221) operation on a derived class of the **DDS_TypeSupport** (p. 214) interface.

MT Safety:

UNSAFE. It is not safe to create a topic while another thread is trying to lookup that topic description with **DDS_DomainParticipant_lookup_topicdescription** (p. 127).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>topic_name</i>	<< <i>in</i> >> (p. 1013) Name for the new topic, must not exceed 255 characters. Cannot be NULL.
<i>type_name</i>	<< <i>in</i> >> (p. 1013) The type to which the new DDS_Topic (p. 175) will be bound. Cannot be NULL.
<i>library_name</i>	<< <i>in</i> >> (p. 1013) Library name containing the XML QoS profile. If <i>library_name</i> is null RTI Connex will use the default library (see DDS_DomainParticipant_set_default_library (p. 101)).
<i>profile_name</i>	<< <i>in</i> >> (p. 1013) XML QoS Profile name. If <i>profile_name</i> is null RTI Connex will use the default profile (see DDS_DomainParticipant_set_default_profile (p. 102)).
<i>listener</i>	<< <i>in</i> >> (p. 1013). Listener to be attached to the newly created DDS_Topic (p. 175).
<i>mask</i>	<< <i>in</i> >> (p. 1013). Changes of communication status to be invoked on the listener. See DDS_StatusMask (p. 1226).

Returns

newly created topic, or NULL on failure

See also

Specifying QoS on entities (p. 1244) for information on setting QoS before entity creation

DDS_TopicQos (p. 2042) for rules on consistency among QoS

DDS_DomainParticipant_create_topic (p. 115)

DDS_DomainParticipant_get_default_topic_qos (p. 86)

DDS_Topic_set_listener (p. 199)

5.4.4.50 DDS_DomainParticipant_delete_topic()

```
DDS_ReturnCode_t DDS_DomainParticipant_delete_topic (
    DDS_DomainParticipant * self,
    DDS_Topic * topic )
```

Deletes a **DDS_Topic** (p. 175).

Precondition

If the **DDS_Topic** (p. 175) does not belong to the application's **DDS_DomainParticipant** (p. 77), this operation fails with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

Make sure no objects are using the topic. More specifically, there must be no existing **DDS_DataReader** (p. 615), **DDS_DataWriter** (p. 485), **DDS_ContentFilteredTopic** (p. 177), or **DDS_MultiTopic** (p. 185) objects belonging to the same **DDS_DomainParticipant** (p. 77) that are using the **DDS_Topic** (p. 175). If `delete_topic` is called on a **DDS_Topic** (p. 175) with any of these existing objects attached to it, it will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

Postcondition

Listener installed on the **DDS_Topic** (p. 175) will not be called after this function 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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>topic</i>	<< <i>in</i> >> (p. 1013) DDS_Topic (p. 175) to be deleted.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222)

5.4.4.51 **DDS_DomainParticipant_create_contentfilteredtopic()**

```

DDS_ContentFilteredTopic * DDS_DomainParticipant_create_contentfilteredtopic (
    DDS_DomainParticipant * self,
    const char * name,
    DDS_Topic * related_topic,
    const char * filter_expression,
    const struct DDS_StringSeq * expression_parameters )

```

Creates a **DDS_ContentFilteredTopic** (p. 177), that can be used to do content-based subscriptions.

The **DDS_ContentFilteredTopic** (p. 177) only relates to samples published under that **DDS_Topic** (p. 175), 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. 738) describes the syntax of *filter_expression* and *expression_parameters*.

Precondition

The application is not allowed to create two **DDS_ContentFilteredTopic** (p. 177) objects with the same *topic_name* attached to the same **DDS_DomainParticipant** (p. 77). If the application attempts this, this function will fail and returns NULL.

If *related_topic* does not belong to this **DDS_DomainParticipant** (p. 77), 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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>name</i>	<< <i>in</i> >> (p. 1013) Name for the new content filtered topic, must not exceed 255 characters. Cannot be NULL.
<i>related_topic</i>	<< <i>in</i> >> (p. 1013) DDS_Topic (p. 175) to be filtered. Cannot be NULL.
<i>filter_expression</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL
<i>expression_parameters</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL. 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 **DDS_ContentFilteredTopic** (p. 177), or NULL on failure

5.4.4.52 DDS_DomainParticipant_create_contentfilteredtopic_with_filter()

```
DDS_ContentFilteredTopic * DDS_DomainParticipant_create_contentfilteredtopic_with_filter (
    DDS_DomainParticipant * self,
    const char * name,
    const DDS_Topic * related_topic,
    const char * filter_expression,
    const struct DDS_StringSeq * expression_parameters,
    const char * filter_name )
```

<<**extension**>> (p. 1012) Creates a **DDS_ContentFilteredTopic** (p.177) using the specified filter to do content-based subscriptions.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>name</i>	<< in >> (p. 1013) Name for the new content filtered topic. Cannot exceed 255 characters. Cannot be NULL.
<i>related_topic</i>	<< in >> (p. 1013) DDS_Topic (p. 175) to be filtered. Cannot be NULL.
<i>filter_expression</i>	<< in >> (p. 1013) Cannot be NULL.
<i>expression_parameters</i>	<< in >> (p. 1013) Cannot be NULL. . An empty sequence must be used if the filter expression does not contain any parameters. Length of the sequence cannot be greater than 100.
<i>filter_name</i>	<< in >> (p. 1013) Name of content filter to use. Must previously have been registered with DDS_DomainParticipant_register_contentfilter (p. 120) on the same DDS_DomainParticipant (p. 77). Cannot be NULL. Builtin filter names are DDS_SQLFILTER_NAME (p. 163) and DDS_STRINGMATCHFILTER_NAME (p. 164)

Returns

newly created **DDS_ContentFilteredTopic** (p. 177), or NULL on failure

5.4.4.53 DDS_DomainParticipant_delete_contentfilteredtopic()

```
DDS_ReturnCode_t DDS_DomainParticipant_delete_contentfilteredtopic (
    DDS_DomainParticipant * self,
    DDS_ContentFilteredTopic * a_contentfilteredtopic )
```

Deletes a **DDS_ContentFilteredTopic** (p. 177).

Precondition

The deletion of a **DDS_ContentFilteredTopic** (p. 177) is not allowed if there are any existing **DDS_DataReader** (p. 615) objects that are using the **DDS_ContentFilteredTopic** (p. 177). If the operation is called on a **DDS_ContentFilteredTopic** (p. 177) with existing **DDS_DataReader** (p. 615) objects attached to it, it will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

The **DDS_ContentFilteredTopic** (p. 177) must be created by this **DDS_DomainParticipant** (p. 77), or else this operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>a_contentfilteredtopic</i>	<< <i>in</i> >> (p. 1013)

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222)

5.4.4.54 DDS_DomainParticipant_register_contentfilter()

```
DDS_ReturnCode_t DDS_DomainParticipant_register_contentfilter (
    DDS_DomainParticipant * self,
    const char * filter_name,
    const struct DDS_ContentFilter * contentfilter )
```

<<**extension**>> (p. 1012) Register a content filter which can be used to create a **DDS_ContentFilteredTopic** (p. 177).

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 **DDS_ContentFilteredTopic** (p. 177) and the corresponding **DDS_DataReader** (p. 615).
- 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 **DDS_DomainParticipant** (p. 77).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>filter_name</i>	<< <i>in</i> >> (p. 1013) Name of the filter. The name must be unique within the DDS_DomainParticipant (p. 77) and must not exceed 255 characters. Cannot be NULL.
<i>contentfilter</i>	<< <i>in</i> >> (p. 1013) Content filter to be registered. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DomainParticipant_unregister_contentfilter (p. 121)

5.4.4.55 DDS_DomainParticipant_lookup_contentfilter()

```
void * DDS_DomainParticipant_lookup_contentfilter (
    DDS_DomainParticipant * self,
    const char * filter_name )
```

<<*extension*>> (p. 1012) Lookup a content filter previously registered with **DDS_DomainParticipant_register_contentfilter** (p. 120).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>filter_name</i>	<< <i>in</i> >> (p. 1013) Name of the filter. Cannot be NULL.

Returns

NULL if the given filter_name has not been previously registered to the **DDS_DomainParticipant** (p. 77) with **DDS_DomainParticipant_register_contentfilter** (p. 120). Otherwise, return the filter_data that has been previously registered with the given filter_name.

See also

DDS_DomainParticipant_register_contentfilter (p. 120)

5.4.4.56 DDS_DomainParticipant_unregister_contentfilter()

```
DDS_ReturnCode_t DDS_DomainParticipant_unregister_contentfilter (
    DDS_DomainParticipant * self,
    const char * filter_name )
```

<<**extension**>> (p. 1012) Unregister a content filter previously registered with **DDS_DomainParticipant_register_contentfilter** (p. 120).

A `filter_name` can be unregistered only if it has been previously registered to the **DDS_DomainParticipant** (p. 77) with **DDS_DomainParticipant_register_contentfilter** (p. 120).

The unregistration of filter is not allowed if there are any existing **DDS_ContentFilteredTopic** (p. 177) objects that are using the filter. If the operation is called on a filter with existing **DDS_ContentFilteredTopic** (p. 177) objects attached to it, this operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

If there are still existing discovered **DDS_DataReader** (p. 615) s with the same `filter_name` and the filter's compile method of the filter have previously been called on the discovered **DDS_DataReader** (p. 615) s, `finalize` method of the filter will be called on those discovered **DDS_DataReader** (p. 615) s before the content filter is unregistered. This means filtering will now be performed on the application that is creating the **DDS_DataReader** (p. 615).

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>filter_name</i>	<< in >> (p. 1013) Name of the filter. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222)

See also

DDS_DomainParticipant_register_contentfilter (p. 120)

5.4.4.57 DDS_DomainParticipant_create_multitopic()

```
DDS_MultiTopic * DDS_DomainParticipant_create_multitopic (
    DDS_DomainParticipant * self,
    const char * name,
    const char * type_name,
    const char * subscription_expression,
    const struct DDS_StringSeq * expression_parameters )
```

[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 **DDS_Topic** (p. 175) are specified using the `subscription_expression` and `expression_parameters` arguments.

Queries and Filters Syntax (p. 738) describes the syntax of `subscription_expression` and `expression_parameters`.

Precondition

The application is not allowed to create two **DDS_TopicDescription** (p. 175) objects with the same `name` attached to the same **DDS_DomainParticipant** (p. 77). If the application attempts this, this function will fail and return NULL.

Prior to creating a **DDS_MultiTopic** (p. 185), the type must have been registered with RTI Connex. This is done using the **FooTypeSupport_register_type** (p. 221) operation on a derived class of the **DDS_TypeSupport** (p. 214) interface. Otherwise, this function will return NULL.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>name</i>	<< <i>in</i> >> (p. 1013) Name of the newly create DDS_MultiTopic (p. 185). Cannot be NULL.
<i>type_name</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>subscription_expression</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>expression_parameters</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

NULL

5.4.4.58 DDS_DomainParticipant_delete_multitopic()

```
DDS_ReturnCode_t DDS_DomainParticipant_delete_multitopic (
    DDS_DomainParticipant * self,
    DDS_MultiTopic * a_multitopic )
```

[Not supported (optional)] Deletes a **DDS_MultiTopic** (p. 185).

Precondition

The deletion of a **DDS_MultiTopic** (p. 185) is not allowed if there are any existing **DDS_DataReader** (p. 615) objects that are using the **DDS_MultiTopic** (p. 185). If the `delete_multitopic` operation is called on a **DDS_MultiTopic** (p. 185) with existing **DDS_DataReader** (p. 615) objects attached to it, it will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

The **DDS_MultiTopic** (p. 185) must be created by this **DDS_DomainParticipant** (p. 77), or else this operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>a_multitopic</i>	<< <i>in</i> >> (p. 1013)

5.4.4.59 DDS_DomainParticipant_create_flowcontroller()

```
DDS_FlowController * DDS_DomainParticipant_create_flowcontroller (
    DDS_DomainParticipant * self,
    const char * name,
    const struct DDS_FlowControllerProperty_t * prop )
```

<<*extension*>> (p. 1012) Creates a **DDS_FlowController** (p. 559) with the desired property.

The created **DDS_FlowController** (p. 559) is associated with a **DDS_DataWriter** (p. 485) via **DDS_PublishModeQosPolicy::flow_controller_name** (p. 1922). A single FlowController may service multiple DataWriters instances, even if they belong to a different **DDS_Publisher** (p. 443). The `property` determines how the FlowController shapes the network traffic.

Precondition

The specified `property` must be consistent, or the operation will fail and no **DDS_FlowController** (p. 559) will be created.

MT Safety:

UNSAFE. If **DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT** (p. 162) is used for `property`, it is not safe to create the flow controller while another thread may be simultaneously calling **DDS_DomainParticipant_set_default_flowcontroller_property** (p. 99) or trying to lookup that flow controller with **DDS_DomainParticipant_lookup_flowcontroller** (p. 128).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>name</i>	<< <i>in</i> >> (p. 1013) name of the DDS_FlowController (p. 559) to create. A DDS_DataWriter (p. 485) is associated with a DDS_FlowController (p. 559) by name. Limited to 255 characters.
<i>prop</i>	<< <i>in</i> >> (p. 1013) property to be used for creating the new DDS_FlowController (p. 559). The special value DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT (p. 162) can be used to indicate that the DDS_FlowController (p. 559) should be created with the default DDS_FlowControllerProperty_t (p. 1799) set in the DDS_DomainParticipant (p. 77). Cannot be NULL.

Returns

Newly created flow controller object or NULL on failure.

See also

DDS_FlowControllerProperty_t (p. 1799) for rules on consistency among property

DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT (p. 162)

DDS_DomainParticipant_get_default_flowcontroller_property (p. 98)

5.4.4.60 DDS_DomainParticipant_delete_flowcontroller()

```
DDS_ReturnCode_t DDS_DomainParticipant_delete_flowcontroller (
    DDS_DomainParticipant * self,
    DDS_FlowController * fc )
```

<<**extension**>> (p. 1012) Deletes an existing **DDS_FlowController** (p. 559).

Precondition

The **DDS_FlowController** (p. 559) must not have any attached **DDS_DataWriter** (p. 485) objects. If there are any attached **DDS_DataWriter** (p. 485) objects, it will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

The **DDS_FlowController** (p. 559) must have been created by this **DDS_DomainParticipant** (p. 77), or else it will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

Postcondition

The **DDS_FlowController** (p. 559) is deleted if this function 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

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>fc</i>	<< in >> (p. 1013) The DDS_FlowController (p. 559) to be deleted.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

5.4.4.61 DDS_DomainParticipant_get_typecode()

```
const DDS_TypeCode * DDS_DomainParticipant_get_typecode (
    DDS_DomainParticipant * self,
    const char * type_name )
```

<<**extension**>> (p. 1012) Retrieves the **DDS_TypeCode** (p. 2071) of a type registered with the **DDS_DomainParticipant** (p. 77) based on the registration name.

Every data type used in a **DDS_DomainParticipant** (p. 77) has a registered type name which is the name specified at the time the type is registered with the **DDS_DomainParticipant** (p. 77).

This operation retrieves the **DDS_TypeCode** (p. 2071) for the type given the registered type name. The `type_name` argument must correspond to a name used to register a type with the **DDS_DomainParticipant** (p. 77). Otherwise, this method will return NULL.

Type registration in a **DDS_DomainParticipant** (p. 77) is performed by a call to the **FooTypeSupport_register_type** (p. 221) operation on a derived class of the **DDS_TypeSupport** (p. 214) interface.

Type registration might occur directly via an application call to the **FooTypeSupport_register_type** (p. 221) 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 **DDS_TypeSupport** (p. 214) can be created either from a type description found in the XML files, or else by calling the **FooTypeSupport_register_type** (p. 221) operation on a **DDS_TypeSupport** (p. 214) that has been registered with **DDS_DomainParticipantFactory** (p. 32). If a **DDS_TypeSupport** (p. 214) is registered with the **DDS_DomainParticipantFactory** (p. 32) this mechanism takes precedence over the creation of a **DDS_TypeSupport** (p. 214) from a type description in the XML file.

To register a **DDS_TypeSupport** (p. 214) with the **DDS_DomainParticipantFactory** (p. 32) use the operation **DDS_DomainParticipantFactory_register_type_support** (p. 62).

If the **DDS_TypeSupport** (p. 214) is created from a type description found in the XML files, the resulting type support will be a **DDS_DynamicDataTypeSupport** (p. 326). In this case the **DDS_DataWriter** (p. 485) and **DDS_DataReader** (p. 615) uses to write and read data of that type will be the **DDS_DynamicDataWriter** (p. 325) and **DDS_DynamicDataReader** (p. 326), respectively.

Parameters

<i>self</i>	<<in>> (p. 1013) Cannot be NULL.
<i>type_name</i>	<<in>> (p. 1013) Name of the type whose TypeCode is retrieved.

Returns

The TypeCode of a registered type with the **DDS_DomainParticipant** (p. 77) or NULL if no type was registered with the participant under that name.

5.4.4.62 DDS_DomainParticipant_find_topic()

```
DDS_Topic * DDS_DomainParticipant_find_topic (
    DDS_DomainParticipant * self,
    const char * topic_name,
    const struct DDS_Duration_t * timeout )
```

Finds an existing (or ready to exist) **DDS_Topic** (p. 175), based on its name.

This call can be used to block for a specified duration to wait for the **DDS_Topic** (p. 175) to be created.

If the requested **DDS_Topic** (p. 175) already exists, it is returned. Otherwise, `find_topic()` waits until another thread creates it or else returns when the specified timeout occurs.

`find_topic()` is useful when multiple threads are concurrently creating and looking up topics. In that case, one thread can call `find_topic()` 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 **DDS_DomainParticipant_lookup_topicdescription** (p. 127).

The **DDS_DomainParticipant** (p. 77) must already be enabled.

Note: Each **DDS_Topic** (p. 175) obtained by **DDS_DomainParticipant_find_topic** (p. 126) must also be deleted by means of **DDS_DomainParticipant_delete_topic** (p. 117). If **DDS_Topic** (p. 175) is obtained multiple times by means of **DDS_DomainParticipant_find_topic** (p. 126) or **DDS_DomainParticipant_create_topic** (p. 115), it must also be deleted that same number of times using **DDS_DomainParticipant_delete_topic** (p. 117).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>topic_name</i>	<< <i>in</i> >> (p. 1013) Name of the DDS_Topic (p. 175) to search for. Cannot be NULL.
<i>timeout</i>	<< <i>in</i> >> (p. 1013) The time to wait if the DDS_Topic (p. 175) does not exist already. Cannot be NULL.

Returns

the topic, if it exists, or NULL

5.4.4.63 DDS_DomainParticipant_lookup_topicdescription()

```
DDS_TopicDescription * DDS_DomainParticipant_lookup_topicdescription (
    DDS_DomainParticipant * self,
    const char * topic_name )
```

Looks up an existing, locally created **DDS_TopicDescription** (p. 175), based on its name.

DDS_TopicDescription (p. 175) is the base class for **DDS_Topic** (p. 175), **DDS_MultiTopic** (p. 185) and **DDS_ContentFilteredTopic** (p. 177). So you can narrow the **DDS_TopicDescription** (p. 175) returned from this operation to a **DDS_Topic** (p. 175) or **DDS_ContentFilteredTopic** (p. 177) as appropriate.

Unlike **DDS_DomainParticipant_find_topic** (p. 126), which logically returns a new **DDS_Topic** (p. 175) object that must be independently deleted, *this* operation returns a reference to the original local object.

The **DDS_DomainParticipant** (p. 77) does not have to be enabled when you call `lookup_topicdescription()`.

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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>topic_name</i>	<< <i>in</i> >> (p. 1013) Name of DDS_TopicDescription (p. 175) 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.

5.4.4.64 **DDS_DomainParticipant_lookup_flowcontroller()**

```
DDS_FlowController * DDS_DomainParticipant_lookup_flowcontroller (
    DDS_DomainParticipant * self,
    const char * name )
```

<<*extension*>> (p. 1012) Looks up an existing locally-created **DDS_FlowController** (p. 559), based on its name.

Looks up a previously created **DDS_FlowController** (p. 559), including the built-in ones. Once a **DDS_FlowController** (p. 559) 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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>name</i>	<< <i>in</i> >> (p. 1013) Name of DDS_FlowController (p. 559) to search for. Limited to 255 characters. Cannot be NULL.

Returns

The flow controller if it has already been created locally, or NULL otherwise.

5.4.4.65 **DDS_DomainParticipant_get_participant_protocol_status()**

```
DDS_ReturnCode_t DDS_DomainParticipant_get_participant_protocol_status (
    DDS_DomainParticipant * self,
    struct DDS_DomainParticipantProtocolStatus * status )
```

<<*extension*>> (p. 1012) Get the domain participant protocol status for this participant.

This also resets the status so that it is no longer considered changed.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>status</i>	<< <i>inout</i> >> (p. 1013) DDS_DomainParticipantProtocolStatus (p. 1725) to be filled in. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

5.4.4.66 DDS_DomainParticipant_get_builtin_subscriber()

```
DDS_Subscriber * DDS_DomainParticipant_get_builtin_subscriber (
    DDS_DomainParticipant * self )
```

Accesses the **built-in DDS_Subscriber** (p. 572).

Each **DDS_DomainParticipant** (p. 77) contains several built-in **DDS_Topic** (p. 175) objects as well as corresponding **DDS_DataReader** (p. 615) objects to access them. All of these **DDS_DataReader** (p. 615) objects belong to a single built-in **DDS_Subscriber** (p. 572).

The built-in Topics are used to communicate information about other **DDS_DomainParticipant** (p. 77), **DDS_Topic** (p. 175), **DDS_DataReader** (p. 615), and **DDS_DataWriter** (p. 485) 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 **DDS_DomainParticipant** (p. 77) is deleted.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

The built-in **DDS_Subscriber** (p. 572) singleton.

See also

DDS_SubscriptionBuiltinTopicData (p. 2012)

DDS_PublicationBuiltinTopicData (p. 1905)

DDS_ParticipantBuiltinTopicData (p. 1870)

DDS_TopicBuiltinTopicData (p. 2034)

5.4.4.67 DDS_DomainParticipant_get_implicit_publisher()

```
DDS_Publisher * DDS_DomainParticipant_get_implicit_publisher (
    DDS_DomainParticipant * self )
```

<<**extension**>> (p. 1012) Returns the implicit **DDS_Publisher** (p. 443). 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. 161) and no Listener.

This implicit Publisher will be deleted automatically when the following functions are called: **DDS_DomainParticipant_delete_contained_entities** (p. 135), or **DDS_DomainParticipant_delete_publisher** (p. 105) 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.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.

Returns

The implicit publisher

See also

DDS_PUBLISHER_QOS_DEFAULT (p. 161)

DDS_DomainParticipant_create_publisher (p. 103)

5.4.4.68 DDS_DomainParticipant_get_implicit_subscriber()

```
DDS_Subscriber * DDS_DomainParticipant_get_implicit_subscriber (
    DDS_DomainParticipant * self )
```

<<**extension**>> (p. 1012) Returns the implicit **DDS_Subscriber** (p. 572). 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. 162) and no Listener.

This implicit Subscriber will be deleted automatically when the following functions are called: **DDS_DomainParticipant_delete_contained_entities** (p. 135), or **DDS_DomainParticipant_delete_subscriber** (p. 108) 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 **DDS_DomainParticipant_set_default_subscriber_qos** (p. 94).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

The implicit subscriber

See also

DDS_SUBSCRIBER_QOS_DEFAULT (p. 162)

DDS_DomainParticipant_create_subscriber (p. 106)

5.4.4.69 DDS_DomainParticipant_ignore_participant()

```
DDS_ReturnCode_t DDS_DomainParticipant_ignore_participant (
    DDS_DomainParticipant * self,
    const DDS_InstanceHandle_t * handle )
```

Instructs RTI Connex to locally ignore a remote **DDS_DomainParticipant** (p. 77).

From the time of this call onwards, RTI Connex 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 **DDS_DomainParticipant** (p. 77).

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. 1870) to provide access control.

Application data can be associated with a **DDS_DomainParticipant** (p. 77) by means of the **USER_DATA** (p. 1343) 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 **DDS_DomainParticipant** (p. 77) to ignore is identified by the `handle` argument. This `handle` is the one that appears in the **DDS_SampleInfo** (p. 1985) retrieved when reading the data-samples available for the built-in **DDS_↔DataReader** (p. 615) to the **DDS_DomainParticipant** (p. 77) topic. The built-in **DDS_DataReader** (p. 615) is read with the same **FooDataReader_read** (p. 625) and **FooDataReader_take** (p. 626) operations used for any **DDS_DataReader** (p. 615).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>handle</i>	<< <i>in</i> >> (p. 1013) DDS_InstanceHandle_t (p. 214) of the DDS_DomainParticipant (p. 77) to be ignored. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222), **DDS_RETCODE_NOT_ENABLED** (p. 1222)

See also

DDS_ParticipantBuiltinTopicData (p. 1870)

DDS_PARTICIPANT_TOPIC_NAME (p. 1092)

DDS_DomainParticipant_get_builtin_subscriber (p. 129)

5.4.4.70 DDS_DomainParticipant_ignore_topic()

```
DDS_ReturnCode_t DDS_DomainParticipant_ignore_topic (
    DDS_DomainParticipant * self,
    const DDS_InstanceHandle_t * handle )
```

Instructs RTI Connnext to locally ignore a **DDS_Topic** (p. 175).

This means it will locally ignore any publication, or subscription to the **DDS_Topic** (p. 175).

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 **DDS_Topic** (p. 175) to ignore is identified by the `handle` argument. This is the handle of a **DDS_Topic** (p. 175) that appears in the **DDS_SampleInfo** (p. 1985) retrieved when reading data samples from the built-in **DDS_DataReader** (p. 615) for the **DDS_Topic** (p. 175).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>handle</i>	<< <i>in</i> >> (p. 1013) Handle of the DDS_Topic (p. 175) to be ignored. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222)

See also

DDS_TopicBuiltinTopicData (p. 2034)

DDS_TOPIC_TOPIC_NAME (p. 1094)

DDS_DomainParticipant_get_builtin_subscriber (p. 129)

5.4.4.71 DDS_DomainParticipant_ignore_publication()

```
DDS_ReturnCode_t DDS_DomainParticipant_ignore_publication (
    DDS_DomainParticipant * self,
    const DDS_InstanceHandle_t * handle )
```

Instructs RTI Connex to locally ignore a publication.

A publication is defined by the association of a topic name, user data, and partition set on the **DDS_Publisher** (p. 443) (see **DDS_PublicationBuiltinTopicData** (p. 1905)). After this call, any data written by that publication's **DDS_DataWriter** (p. 485) 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. 1985) retrieved when reading data samples from the built-in **DDS_DataReader** (p. 615) for the publication topic.
- To ignore a *local* DataWriter, the `handle` can be obtained by calling **DDS_Entity_get_instance_handle** (p. 1366) for the local DataWriter.

There is no way to reverse this operation.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>handle</i>	<< <i>in</i> >> (p. 1013) Handle of the DDS_DataWriter (p. 485) to be ignored. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222)

See also

DDS_PublicationBuiltinTopicData (p. 1905)
DDS_PUBLICATION_TOPIC_NAME (p. 1096)
DDS_DomainParticipant_get_builtin_subscriber (p. 129)

5.4.4.72 DDS_DomainParticipant_ignore_subscription()

```
DDS_ReturnCode_t DDS_DomainParticipant_ignore_subscription (
    DDS_DomainParticipant * self,
    const DDS_InstanceHandle_t * handle )
```

Instructs RTI Connex to locally ignore a subscription.

A subscription is defined by the association of a topic name, user data, and partition set on the **DDS_Subscriber** (p. 572) (see **DDS_SubscriptionBuiltinTopicData** (p.2012)). After this call, any data received related to that subscription's **DDS_DataReader** (p.615) 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.1985) retrieved when reading data samples from the built-in **DDS_DataReader** (p.615) for the subscription topic.
- To ignore a *local* DataReader, the `handle` can be obtained by calling **DDS_Entity_get_instance_handle** (p.1366) for the local DataReader.

There is no way to reverse this operation.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>handle</i>	<< <i>in</i> >> (p. 1013) Handle of the DDS_DataReader (p.615) to be ignored. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p.1221), **DDS_RETCODE_OUT_OF_RESOURCES** (p.1222) or **DDS_↔ RETCODE_NOT_ENABLED** (p.1222)

See also

DDS_SubscriptionBuiltinTopicData (p.2012)
DDS_SUBSCRIPTION_TOPIC_NAME (p.1098)
DDS_DomainParticipant_get_builtin_subscriber (p.129)

5.4.4.73 DDS_DomainParticipant_banish_ignored_participants()

```
DDS_ReturnCode_t DDS_DomainParticipant_banish_ignored_participants (
    DDS_DomainParticipant * self )
```

<<*extension*>> (p. 1012) Prevents ignored remote DomainParticipants from receiving traffic from the local **DDS_↔ DomainParticipant** (p.77).

This method complements **DDS_DomainParticipant_ignore_participant** (p.131): `ignore_participant` prevents the local **DDS_DomainParticipant** (p.77) 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.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222), **DDS_RETCODE_NOT_ENABLED** (p. 1222)

See also

DDS_DomainParticipant_ignore_participant (p. 131)

5.4.4.74 DDS_DomainParticipant_get_domain_id()

```
DDS_DomainId_t DDS_DomainParticipant_get_domain_id (
    DDS_DomainParticipant * self )
```

Get the unique domain identifier.

This operation retrieves the domain id used to create the **DDS_DomainParticipant** (p. 77). The domain id identifies the DDS domain to which the **DDS_DomainParticipant** (p. 77) belongs. Each DDS domain represents a separate data 'communication plane' isolated from other domains.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

the unique `domainId` that was used to create the domain

See also

DDS_DomainParticipantFactory_create_participant (p. 41)

DDS_DomainParticipantFactory_create_participant_with_profile (p. 43)

5.4.4.75 DDS_DomainParticipant_delete_contained_entities()

```
DDS_ReturnCode_t DDS_DomainParticipant_delete_contained_entities (
    DDS_DomainParticipant * self )
```

Delete all the entities that were created by means of the "create" operations on the **DDS_DomainParticipant** (p. 77).

This operation deletes all contained **DDS_Publisher** (p. 443) (including an implicit Publisher, if one exists), **DDS_Subscriber** (p. 572) (including implicit Subscriber), **DDS_Topic** (p. 175), **DDS_ContentFilteredTopic** (p. 177), and **DDS_MultiTopic** (p. 185) objects.

Prior to deleting each contained entity, this operation will recursively call the corresponding `delete_contained_entities()` operation on each contained entity (if applicable). This pattern is applied recursively. In this manner the operation `delete_contained_entities()` on the **DDS_DomainParticipant** (p. 77) will end up recursively deleting all the entities contained in the **DDS_DomainParticipant** (p. 77), including the **DDS_DataWriter** (p. 485), **DDS_DataReader** (p. 615), as well as the **DDS_QueryCondition** (p. 699), **DDS_ReadCondition** (p. 694), and **DDS_TopicQuery** (p. 707) objects belonging to the contained **DDS_DataReader** (p. 615).

The operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) if any of the contained entities is in a state where it cannot be deleted .

If `delete_contained_entities()` completes successfully, the application may delete the **DDS_DomainParticipant** (p. 77).

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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

Examples

HelloWorld_publisher.c, and **HelloWorld_subscriber.c**.

5.4.4.76 DDS_DomainParticipant_assert_liveliness()

```
DDS_ReturnCode_t DDS_DomainParticipant_assert_liveliness (
    DDS_DomainParticipant * self )
```

Manually asserts the liveliness of this **DDS_DomainParticipant** (p. 77).

This is used in combination with the **DDS_LivelinessQosPolicy** (p. 1823) to indicate to RTI Connext that the entity remains active.

You need to use this operation if the **DDS_DomainParticipant** (p. 77) contains **DDS_DataWriter** (p. 485) entities with the **DDS_LivelinessQosPolicy::kind** (p. 1825) set to **DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS** (p. 1297) and it only affects the liveliness of those **DDS_DataWriter** (p. 485) entities. Otherwise, it has no effect.

Note: writing data via the **FooDataWriter_write** (p. 496) or **FooDataWriter_write_w_timestamp** (p. 500) operation asserts liveliness on the **DDS_DataWriter** (p. 485) itself and its **DDS_DomainParticipant** (p. 77). Consequently the use of `assert_liveliness()` is only needed if the application is not writing data regularly.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_NOT_ENABLED** (p. 1222)

See also

DDS_LivelinessQosPolicy (p. 1823)

5.4.4.77 DDS_DomainParticipant_get_publishers()

```
DDS_ReturnCode_t DDS_DomainParticipant_get_publishers (
    DDS_DomainParticipant * self,
    struct DDS_PublisherSeq * publishers )
```

<<*extension*>> (p. 1012) 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. 1222).

MT Safety:

Safe.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>publishers</i>	<< <i>inout</i> >> (p. 1013) a PublisherSeq object where the set or list of publishers will be returned

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

5.4.4.78 DDS_DomainParticipant_get_subscribers()

```
DDS_ReturnCode_t DDS_DomainParticipant_get_subscribers (
    DDS_DomainParticipant * self,
    struct DDS_SubscriberSeq * subscribers )
```

<<*extension*>> (p. 1012) 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. 1222).

MT Safety:

Safe.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>subscribers</i>	<< <i>inout</i> >> (p. 1013) a SubscriberSeq object where the set or list of subscribers will be returned

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

5.4.4.79 DDS_DomainParticipant_get_current_time()

```
DDS_ReturnCode_t DDS_DomainParticipant_get_current_time (
    DDS_DomainParticipant * self,
    struct DDS_Time_t * current_time )
```

Returns the current value of the time.

The current value of the time that RTI Connext uses to time-stamp **DDS_DataWriter** (p. 485) and to set the reception-timestamp for the data updates that it receives.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>current_time</i>	<< <i>inout</i> >> (p. 1013) Current time to be filled up. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.4.4.80 DDS_DomainParticipant_contains_entity()

```
DDS_Boolean DDS_DomainParticipant_contains_entity (
    DDS_DomainParticipant * self,
    const DDS_InstanceHandle_t * a_handle )
```

Completes successfully with **DDS_BOOLEAN_TRUE** (p. 1201) if the referenced **DDS_Entity** (p. 1360) is contained by the **DDS_DomainParticipant** (p. 77).

This operation checks whether or not the given `a_handle` represents an **DDS_Entity** (p. 1360) that was created from the **DDS_DomainParticipant** (p. 77). The containment applies recursively. That is, it applies both to entities (**DDS_↵TopicDescription** (p. 175), **DDS_Publisher** (p. 443), or **DDS_Subscriber** (p. 572)) created directly using the **DDS_↵DomainParticipant** (p. 77) as well as entities created using a contained **DDS_Publisher** (p. 443), or **DDS_Subscriber** (p. 572) as the factory, and so forth.

The `instance` handle for an **DDS_Entity** (p. 1360) may be obtained from built-in topic data, from various statuses, or from the operation **DDS_Entity_get_instance_handle** (p. 1366).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>a_handle</i>	<< <i>in</i> >> (p. 1013) DDS_InstanceHandle_t (p. 214) of the DDS_Entity (p. 1360) to be checked.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if **DDS_Entity** (p. 1360) is contained by the **DDS_DomainParticipant** (p. 77), or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.4.4.81 DDS_DomainParticipant_register_durable_subscription()

```
DDS_ReturnCode_t DDS_DomainParticipant_register_durable_subscription (
    DDS_DomainParticipant * self,
    const struct DDS_EndpointGroup_t * group,
    const char * topic_name )
```

<<*extension*>> (p. 1012) Registers a Durable Subscription on the specified **DDS_Topic** (p. 175) on all Persistence Services.

If you need to receive all samples published on a **DDS_Topic** (p. 175), including the ones published while a **DDS_↵DataReader** (p. 615) 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 **DDS_Topic** (p. 175) 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 **DDS_Topic** (p. 175), the Persistence Service will implicitly delete the previous Durable Subscription and create a new one on the new **DDS_Topic** (p. 175).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>group</i>	<< <i>in</i> >> (p. 1013) DDS_EndpointGroup_t (p. 1784) The Durable Subscription name and quorum.
<i>topic_name</i>	<< <i>in</i> >> (p. 1013) The topic name for which the Durable Subscription is created.

Returns

One of the **Standard Return Codes** (p. 1221)

5.4.4.82 DDS_DomainParticipant_delete_durable_subscription()

```
DDS_ReturnCode_t DDS_DomainParticipant_delete_durable_subscription (
    DDS_DomainParticipant * self,
    const struct DDS_EndpointGroup_t * group )
```

<<**extension**>> (p. 1012) 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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>group</i>	<< <i>in</i> >> (p. 1013) DDS_EndpointGroup_t (p. 1784) specifying the Durable Subscription name. Quorum is not required for this operation.

Returns

One of the **Standard Return Codes** (p. 1221)

5.4.4.83 DDS_DomainParticipant_resume_endpoint_discovery()

```
DDS_ReturnCode_t DDS_DomainParticipant_resume_endpoint_discovery (
    DDS_DomainParticipant * self,
    const DDS_InstanceHandle_t * remote_participant_handle )
```

<<**extension**>> (p. 1012) Initiates endpoint discovery with the specified remote **DDS_DomainParticipant** (p. 77).

If the operation returns **DDS_RETCODE_OK** (p. 1222), the **DDS_DomainParticipant** (p. 77) will initiate endpoint discovery with the remote **DDS_DomainParticipant** (p. 77) provided as a parameter.

When **DDS_DiscoveryQosPolicy::enable_endpoint_discovery** (p.1718) is set to **DDS_BOOLEAN_FALSE** (p.1201), this operation allows the RTI Connex 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. 1718) is set to **DDS_BOOLEAN_TRUE** (p. 1201), endpoint discovery will automatically occur for every discovered **DDS_DomainParticipant** (p. 77). In this case, invoking this operation will have no effect and will return **DDS_RETCODE_OK** (p. 1222).

When **DDS_DiscoveryQosPolicy::enable_endpoint_discovery** (p.1718) is set to **DDS_BOOLEAN_FALSE** (p. 1201), you have two options after a remote **DDS_DomainParticipant** (p. 77) is discovered:

- Call this operation to enable endpoint discovery. After invoking this operation, the **DDS_DomainParticipant** (p. 77) will start to exchange endpoint information so that matching and communication can occur with the remote **DDS_DomainParticipant** (p. 77).
- Call the **DDS_DomainParticipant_ignore_participant** (p. 131) operation to permanently ignore endpoint discovery with the remote **DDS_DomainParticipant** (p. 77).

Setting **DDS_DiscoveryQosPolicy::enable_endpoint_discovery** (p. 1718) to **DDS_BOOLEAN_FALSE** (p. 1201) enables application-level authentication use cases, in which a **DDS_DomainParticipant** (p. 77) will initiate endpoint discovery with a remote **DDS_DomainParticipant** (p. 77) after successful authentication at the application level.

The `remote_participant_handle` parameter is the one that appears in the **DDS_SampleInfo** (p.1985) retrieved when reading the data samples available for the built-in **DDS_ParticipantBuiltinTopicDataDataReader** (p. 1092).

If the specified remote **DDS_DomainParticipant** (p. 77) is not in the database of discovered DomainParticipants or has been previously ignored, this operation will fail with **DDS_RETCODE_ERROR** (p. 1222).

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. 1222).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>remote_participant_handle</i>	<< <i>in</i> >> (p. 1013) Handle of a discovered DDS_DomainParticipant (p. 77) for which endpoint discovery is to be resumed. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_NOT_ENABLED** (p. 1222)

See also

DDS_DiscoveryQosPolicy (p. 1715)

5.4.4.84 DDS_DomainParticipant_set_dns_tracker_polling_period()

```
DDS_ReturnCode_t DDS_DomainParticipant_set_dns_tracker_polling_period (
    DDS_DomainParticipant * self,
    const struct DDS_Duration_t * polling_period )
```

<<**extension**>> (p. 1012) 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. 1208) is also accepted as a valid value. If the duration is set to **DDS_DURATION_INFINITE** (p. 1208), 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. 1711) when it is retrieved with **DDS_DomainParticipant_get_qos()** (p. 152).

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>polling_period</i>	<< in >> (p. 1013) Duration that is set as the polling period for the DNS tracker.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DiscoveryConfigQosPolicy::dns_tracker_polling_period (p. 1711)

DDS_DomainParticipant_add_peer (p. 149)

5.4.4.85 DDS_DomainParticipant_get_dns_tracker_polling_period()

```
DDS_ReturnCode_t DDS_DomainParticipant_get_dns_tracker_polling_period (
    DDS_DomainParticipant * self,
    struct DDS_Duration_t * polling_period )
```

<<**extension**>> (p. 1012) 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. 1711). If the value returned is **DDS_DURATION_INFINITE** (p. 1208), the DNS tracker is disabled.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>polling_period</i>	<< out >> (p. 1013) Duration that the API populates with the period of the DNS tracker.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DiscoveryConfigQosPolicy::dns_tracker_polling_period (p. 1711)

5.4.4.86 DDS_DomainParticipant_get_discovered_participants()

```
DDS_ReturnCode_t DDS_DomainParticipant_get_discovered_participants (
    DDS_DomainParticipant * self,
    struct DDS_InstanceHandleSeq * participant_handles )
```

Returns a list of discovered **DDS_DomainParticipant** (p. 77) entities.

This operation retrieves the list of **DDS_DomainParticipant** (p. 77) entities that have been discovered in the domain and that the application has not indicated should be "ignored" by means of the **DDS_DomainParticipant_ignore_participant** (p. 131) operation. When using **DDS_DISCOVERYCONFIG_BUILTIN_SPDP2** (p. 1281), this list only includes **DDS_DomainParticipant** (p. 77) entities that the application has received configuration information from.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>participant_handles</i>	<< <i>inout</i> >> (p. 1013) DDS_InstanceHandleSeq (p. 1809) to be filled with handles of the discovered DDS_DomainParticipant (p. 77) entities.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_NOT_ENABLED** (p. 1222)

5.4.4.87 DDS_DomainParticipant_get_discovered_participants_from_subject_name()

```
DDS_ReturnCode_t DDS_DomainParticipant_get_discovered_participants_from_subject_name (
    DDS_DomainParticipant * self,
    struct DDS_InstanceHandleSeq * participant_handles,
    const char * subject_name )
```

<<*extension*>> (p. 1012) Returns a list of discovered **DDS_DomainParticipant** (p. 77) entities that have the given **DDS_EntityNameQosPolicy::name** (p. 1790).

This operation retrieves the same list as **DDS_DomainParticipant_get_discovered_participants** (p. 144), except this list contains only the participants that have the given **DDS_EntityNameQosPolicy::name** (p. 1790).

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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>participant_handles</i>	<< <i>inout</i> >> (p. 1013) DDS_InstanceHandleSeq (p. 1809) to be filled with handles of the discovered DDS_DomainParticipant (p. 77) entities.
<i>subject_name</i>	<< <i>in</i> >> (p. 1013) The DDS_EntityNameQosPolicy::name (p. 1790) by which to filter the list.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_NOT_ENABLED** (p. 1222)

5.4.4.88 DDS_DomainParticipant_get_discovered_participant_data()

```
DDS_ReturnCode_t DDS_DomainParticipant_get_discovered_participant_data (
    DDS_DomainParticipant * self,
    struct DDS_ParticipantBuiltinTopicData * participant_data,
    const DDS_InstanceHandle_t * participant_handle )
```

Returns **DDS_ParticipantBuiltinTopicData** (p. 1870) for the specified **DDS_DomainParticipant** (p. 77).

This operation retrieves information on a **DDS_DomainParticipant** (p. 77) 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 **DDS_DomainParticipant_ignore_participant** (p. 131) operation.

The *participant_handle* must correspond to such a DomainParticipant. Otherwise, the operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

Use the operation **DDS_DomainParticipant_get_discovered_participants** (p. 144) to find the **DDS_DomainParticipant** (p. 77) entities that are currently discovered.

MT Safety:

Safe.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>participant_data</i>	<< <i>inout</i> >> (p. 1013) DDS_ParticipantBuiltinTopicData (p. 1870) to be filled in with the data for the specified DDS_DomainParticipant (p. 77).
<i>participant_handle</i>	<< <i>in</i> >> (p. 1013) DDS_InstanceHandle_t (p. 214) of DDS_DomainParticipant (p. 77).

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222)

See also

DDS_ParticipantBuiltinTopicData (p. 1870)

DDS_DomainParticipant_get_discovered_participants (p. 144)

5.4.4.89 DDS_DomainParticipant_get_discovered_participant_subject_name()

```
DDS_ReturnCode_t DDS_DomainParticipant_get_discovered_participant_subject_name (
    DDS_DomainParticipant * self,
    char * subject_name,
    DDS_UnsignedLong * subject_name_size,
    const DDS_InstanceHandle_t * participant_handle )
```

<<*extension*>> (p. 1012) Returns **DDS_EntityNameQosPolicy::name** (p. 1790) for the specified **DDS_DomainParticipant** (p. 77).

This operation retrieves the **DDS_EntityNameQosPolicy::name** (p. 1790) of a **DDS_DomainParticipant** (p. 77) 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 **DDS_DomainParticipant_ignore_participant** (p. 131) operation.

The `participant_handle` must correspond to such a DomainParticipant. If the `participant_handle` is **DDS_HANDLE NIL** (p. 228) or is not a valid **DDS_InstanceHandle_t** (p. 214) for a DomainParticipant, then the operation will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 1222). 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. 1222).

Use the operation **DDS_DomainParticipant_get_discovered_participants** (p. 144) to find the **DDS_DomainParticipant** (p. 77) 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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>subject_name</i>	<< <i>out</i> >> (p. 1013) The char buffer that will be used to store the DDS_EntityNameQosPolicy::name (p. 1790) of the discovered DDS_DomainParticipant (p. 77). If NULL, this function will return the required length of this buffer through the <code>subject_name_size</code> parameter.
<i>subject_name_size</i>	<< <i>inout</i> >> (p. 1013) 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. 1222). If the DDS_EntityNameQosPolicy::name

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222)

See also

DDS_EntityNameQosPolicy::name (p. 1790)

DDS_DomainParticipant_get_discovered_participants (p. 144)

5.4.4.90 DDS_DomainParticipant_get_discovered_topics()

```
DDS_ReturnCode_t DDS_DomainParticipant_get_discovered_topics (
    DDS_DomainParticipant * self,
    struct DDS_InstanceHandleSeq * topic_handles )
```

Returns list of discovered **DDS_Topic** (p. 175) objects.

This operation retrieves the list of **DDS_Topic** (p. 175) s that have been discovered in the domain and that the application has not indicated should be "ignored" by means of the **DDS_DomainParticipant_ignore_topic** (p. 132) operation.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>topic_handles</i>	<< <i>inout</i> >> (p. 1013) DDS_InstanceHandleSeq (p. 1809) to be filled with handles of the discovered DDS_Topic (p. 175) objects

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_NOT_ENABLED** (p. 1222)

5.4.4.91 DDS_DomainParticipant_get_discovered_topic_data()

```
DDS_ReturnCode_t DDS_DomainParticipant_get_discovered_topic_data (
    DDS_DomainParticipant * self,
    struct DDS_TopicBuiltinTopicData * topic_data,
    const DDS_InstanceHandle_t * topic_handle )
```

Returns **DDS_TopicBuiltinTopicData** (p. 2034) for the specified **DDS_Topic** (p. 175).

This operation retrieves information on a **DDS_Topic** (p. 175) that has been discovered by the local Participant and must not have been "ignored" by means of the **DDS_DomainParticipant_ignore_topic** (p. 132) operation.

The `topic_handle` must correspond to such a topic. Otherwise, the operation will fail with **DDS_RETCODE_↵PRECONDITION_NOT_MET** (p. 1222).

This call is not supported for remote topics. If a remote `topic_handle` is used, the operation will fail with **DDS_↵RETCODE_UNSUPPORTED** (p. 1222).

Use the operation **DDS_DomainParticipant_get_discovered_topics** (p. 147) to find the topics that are currently discovered.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>topic_data</i>	<< <i>inout</i> >> (p. 1013) DDS_TopicBuiltinTopicData (p. 2034) to be filled with the specified DDS_Topic (p. 175)'s data.
<i>topic_handle</i>	<< <i>in</i> >> (p. 1013) DDS_InstanceHandle_t (p. 214) of DDS_Topic (p. 175).

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) or **DDS_↵RETCODE_NOT_ENABLED** (p. 1222)

See also

DDS_TopicBuiltinTopicData (p. 2034)

DDS_DomainParticipant_get_discovered_topics (p. 147)

5.4.4.92 DDS_DomainParticipant_take_discovery_snapshot()

```
DDS_ReturnCode_t DDS_DomainParticipant_take_discovery_snapshot (
    DDS_DomainParticipant * self,
    const char * file_name )
```

Take a snapshot of the remote participants discovered by a local one.

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="udp4://192.168.1.170:7411"
```

Precondition

`self` is not NULL.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) The local participant.
<i>file_name</i>	<< <i>in</i> >> (p. 1013) Name of the file where snapshot should be printed. If NULL, the snapshot will be printed through NDDS_Config_Logger (p. 2116). Otherwise, the snapshot will be printed in the file specified by <i>file_name</i> .

Returns

One of the **Standard Return Codes** (p. 1221).

5.4.4.93 DDS_DomainParticipant_add_peer()

```
DDS_ReturnCode_t DDS_DomainParticipant_add_peer (
    DDS_DomainParticipant * self,
    const char * peer_desc_string )
```

<<*extension*>> (p. 1012) Attempt to contact one or more additional peer participants.

Add the given peer description to the list of peers with which this **DDS_DomainParticipant** (p. 77) will try to communicate.

This function may be called at any time after this **DDS_DomainParticipant** (p. 77) has been created (before or after it has been enabled).

If this function is called after **DDS_Entity_enable** (p. 1363), an attempt will be made to contact the new peer(s) immediately.

If this function 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. 1716); the first attempted contact will take place after this **DDS_DomainParticipant** (p. 77) 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. 1718) is set to **DDS_BOOLEAN_TRUE** (p. 1201)). 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 **DDS_DomainParticipant** (p. 77) *attempts* to discover, use the built-in participant topic: **DDS_PARTICIPANT_TOPIC_NAME** (p. 1092).

To remove specific peer locators, you may use **DDS_DomainParticipant_remove_peer** (p. 150). If a peer is removed, the add_peer operation will add it back to the list of peers.

To stop communicating with a peer **DDS_DomainParticipant** (p. 77) that has been discovered, use **DDS_DomainParticipant_ignore_participant** (p. 131).

Adding a peer description with this function has no effect on the **DDS_DiscoveryQosPolicy::initial_peers** (p. 1716) that may be subsequently retrieved with **DDS_DomainParticipant_get_qos()** (p. 152) (because **DDS_DiscoveryQosPolicy** (p. 1715) is immutable).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>peer_desc_string</i>	<< <i>in</i> >> (p. 1013) New peer descriptor to be added. The format is specified in Peer Descriptor Format (p. 1353). Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

Peer Descriptor Format (p. 1353)
DDS_DiscoveryQosPolicy::initial_peers (p. 1716)
DDS_PARTICIPANT_TOPIC_NAME (p. 1092)
DDS_DomainParticipant_get_builtin_subscriber (p. 129)

5.4.4.94 DDS_DomainParticipant_remove_peer()

```
DDS_ReturnCode_t DDS_DomainParticipant_remove_peer (
    DDS_DomainParticipant * self,
    const char * peer_desc_string )
```

<<*extension*>> (p. 1012) Remove one or more peer participants from the list of peers with which this **DDS_DomainParticipant** (p. 77) will try to communicate.

This function may be called any time after this **DDS_DomainParticipant** (p. 77) has been enabled

Calling this function has the following effects:

- If a **DDS_DomainParticipant** (p. 77) was already discovered, it will be locally removed along with all its entities.
- Any further requests coming from a **DDS_DomainParticipant** (p. 77) 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 **DDS_DomainParticipant** (p. 77) will stop sending announcement to those locators.

If remote participants located on a peer that was previously removed are discovered, they will be ignored until the related peer is added back by using **DDS_DomainParticipant_add_peer** (p. 149).

Removing a peer description with this function has no effect on the **DDS_DiscoveryQosPolicy::initial_peers** (p. 1716) that may be subsequently retrieved with **DDS_DomainParticipant_get_qos()** (p. 152) (because **DDS_DiscoveryQosPolicy** (p. 1715) is immutable).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>peer_desc_string</i>	<< <i>in</i> >> (p. 1013) Peer descriptor to be removed. The format is specified in Peer Descriptor Format (p. 1353). Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

Peer Descriptor Format (p. 1353)

DDS_DiscoveryQosPolicy::initial_peers (p. 1716)

DDS_DomainParticipant_add_peer (p. 149)

5.4.4.95 DDS_DomainParticipant_set_qos()

```
DDS_ReturnCode_t DDS_DomainParticipant_set_qos (
    DDS_DomainParticipant * self,
    const struct DDS_DomainParticipantQos * qos )
```

Change the QoS of this DomainParticipant.

The **DDS_DomainParticipantQos::user_data** (p. 1728) and **DDS_DomainParticipantQos::entity_factory** (p. 1728) can be changed. The other policies are immutable.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>in</i> >> (p. 1013) Set of policies to be applied to DDS_DomainParticipant (p. 77). Policies must be consistent. Immutable policies cannot be changed after DDS_DomainParticipant (p. 77) is enabled. The special value DDS_PARTICIPANT_QOS_DEFAULT (p. 65) can be used to indicate that the QoS of the DDS_DomainParticipant (p. 77) should be changed to match the current default DDS_DomainParticipantQos (p. 1726) set in the DDS_DomainParticipantFactory (p. 32). Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 1222) if an immutable policy is changed, **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222) if policies are inconsistent, or **DDS_RETCODE_NOT_ALLOWED_BY_SECURITY** (p. 1222) if policies are not allowed by the Security Plugins

See also

DDS_DomainParticipantQos (p. 1726) for rules on consistency among QoS policies

set_qos (abstract) (p. 1361)

5.4.4.96 DDS_DomainParticipant_set_qos_with_profile()

```
DDS_ReturnCode_t DDS_DomainParticipant_set_qos_with_profile (
    DDS_DomainParticipant * self,
    const char * library_name,
    const char * profile_name )
```

<<**extension**>> (p. 1012) Change the QoS of this domain participant using the input XML QoS profile.

The **DDS_DomainParticipantQos::user_data** (p. 1728) and **DDS_DomainParticipantQos::entity_factory** (p. 1728) can be changed. The other policies are immutable.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) Library name containing the XML QoS profile. If library_name is null RTI Connex will use the default library (see DDS_DomainParticipantFactory_set_default_library (p. 48)).
<i>profile_name</i>	<< in >> (p. 1013) XML QoS Profile name. If profile_name is null RTI Connex will use the default profile (see DDS_DomainParticipantFactory_set_default_profile (p. 49)).

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 1222) if immutable policy is changed, or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222) if policies are inconsistent

See also

DDS_DomainParticipantQos (p. 1726) for rules on consistency among QoS

5.4.4.97 DDS_DomainParticipant_get_qos()

```
DDS_ReturnCode_t DDS_DomainParticipant_get_qos (
    DDS_DomainParticipant * self,
    struct DDS_DomainParticipantQos * qos )
```

Get the participant QoS.

This function may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>inout</i> >> (p. 1013) QoS to be filled up. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

get_qos (abstract) (p. 1362)

5.4.4.98 DDS_DomainParticipant_set_property()

```
DDS_ReturnCode_t DDS_DomainParticipant_set_property (
    DDS_DomainParticipant * self,
    const char * property_name,
    const char * value,
    DDS_Boolean propagate )
```

Set the value for a property that applies to a DomainParticipant.

Warning

This function is not implemented in all APIs and it's intended only for testing purposes. You should use **DDS_↔ DomainParticipant_set_qos** (p. 151) instead.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>property_name</i>	<< <i>in</i> >> (p. 1013). Name of the property that you want to set.
<i>value</i>	<< <i>in</i> >> (p. 1013). New value for the property.
<i>propagate</i>	<< <i>in</i> >> (p. 1013). Indicates if the property will be propagated or not.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataWriter_set_property (p. 552)

DDS_DataReader_set_property (p. 689)

DDS_DomainParticipant_set_qos (p. 151)

5.4.4.99 DDS_DomainParticipant_set_listener()

```
DDS_ReturnCode_t DDS_DomainParticipant_set_listener (
    DDS_DomainParticipant * self,
    const struct DDS_DomainParticipantListener * l,
    DDS_StatusMask mask )
```

Sets the participant listener.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>l</i>	<< <i>in</i> >> (p. 1013) Listener to be installed on the entity.
<i>mask</i>	<< <i>in</i> >> (p. 1013) Changes of communication status to be invoked on the listener. See DDS_StatusMask (p. 1226).

The callback function on the listener cannot be NULL if the corresponding status is turned on in the *mask*.

MT Safety:

Unsafe. This function 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.

Once a participant has been enabled, it is therefore important that the listener not be changed unless some application-specific means are available of ensuring that the old listener cannot be concurrently in use. If this contract is violated, it is possible for the **DDS_Listener::listener_data** (p. 1819) field to have been changed without the callback function pointers having been changed (or vica versa); callback functions may therefore be passed an incorrect **DDS_Listener**↔**::listener_data** (p. 1819) value.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

set_listener (abstract) (p. 1362)

5.4.4.100 DDS_DomainParticipant_get_listener()

```
struct DDS_DomainParticipantListener DDS_DomainParticipant_get_listener (
    DDS_DomainParticipant * self )
```

Get the participant listener.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

Existing listener attached to the **DDS_DomainParticipant** (p. 77).

See also

DDS_DomainParticipant_get_listenerX (p. 156)

get_listener (abstract) (p. 1363)

5.4.4.101 **DDS_DomainParticipant_get_listenerX()**

```
DDS_ReturnCode_t DDS_DomainParticipant_get_listenerX (
    DDS_DomainParticipant * self,
    struct DDS_DomainParticipantListener * listener )
```

<<*extension*>> (p. 1012) Get the participant listener.

An alternative form of **DDS_DomainParticipant_get_listener** (p. 154) that fills in an existing listener structure rather than returning one on the stack.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>listener</i>	<< <i>inout</i> >> (p. 1013) Listener structure to be filled up. Cannot be NULL.

See also

DDS_DomainParticipant_get_listener (p. 154)

get_listener (abstract) (p. 1363)

5.4.4.102 **DDS_DomainParticipant_lookup_publisher_by_name()**

```
DDS_Publisher * DDS_DomainParticipant_lookup_publisher_by_name (
    DDS_DomainParticipant * self,
    const char * publisher_name )
```

<<*extension*>> (p. 1012) Looks up a **DDS_Publisher** (p. 443) by its entity name within this **DDS_DomainParticipant** (p. 77).

Every **DDS_Publisher** (p. 443) in the system has an entity name which is configured and stored in the EntityName policy, **ENTITY_NAME** (p. 1290).

This operation retrieves a **DDS_Publisher** (p. 443) within the **DDS_DomainParticipant** (p. 77) given the entity's name. If there are several **DDS_Publisher** (p. 443) with the same name within the **DDS_DomainParticipant** (p. 77), this function returns the first matching occurrence.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>publisher_name</i>	<< <i>in</i> >> (p. 1013) Entity name of the DDS_Publisher (p. 443).

Returns

The first **DDS_Publisher** (p. 443) found with the specified name or NULL if it is not found.

See also

DDS_DomainParticipant_lookup_datawriter_by_name (p. 158)

5.4.4.103 DDS_DomainParticipant_lookup_subscriber_by_name()

```
DDS_Subscriber * DDS_DomainParticipant_lookup_subscriber_by_name (
    DDS_DomainParticipant * self,
    const char * subscriber_name )
```

<<*extension*>> (p. 1012) Retrieves a **DDS_Subscriber** (p. 572) by its entity name within this **DDS_DomainParticipant** (p. 77).

Every **DDS_Subscriber** (p. 572) in the system has an entity name which is configured and stored in the EntityName policy, **ENTITY_NAME** (p. 1290).

This operation retrieves a **DDS_Subscriber** (p. 572) within the **DDS_DomainParticipant** (p. 77) given the entity's name. If there are several **DDS_Subscriber** (p. 572) with the same name within the **DDS_DomainParticipant** (p. 77), this function returns the first matching occurrence.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>subscriber_name</i>	<< <i>in</i> >> (p. 1013) Entity name of the DDS_Subscriber (p. 572).

Returns

The first **DDS_Subscriber** (p. 572) found with the specified name or NULL if it is not found.

See also

DDS_DomainParticipant_lookup_datareader_by_name (p. 159)

5.4.4.104 DDS_DomainParticipant_lookup_datawriter_by_name()

```
DDS_DataWriter * DDS_DomainParticipant_lookup_datawriter_by_name (
    DDS_DomainParticipant * self,
    const char * datawriter_full_name )
```

<<*extension*>> (p. 1012) Looks up a **DDS_DataWriter** (p. 485) by its entity name within this **DDS_DomainParticipant** (p. 77).

Every **DDS_DataWriter** (p. 485) in the system has an entity name which is configured and stored in the EntityName policy, **ENTITY_NAME** (p. 1290).

Every **DDS_Publisher** (p. 443) in the system has an entity name which is also configured and stored in the EntityName policy.

This operation retrieves a **DDS_DataWriter** (p. 485) within a **DDS_Publisher** (p. 443) given the specified name which encodes both to the **DDS_DataWriter** (p. 485) and the **DDS_Publisher** (p. 443) name.

If there are several **DDS_DataWriter** (p. 485) with the same name within the corresponding **DDS_Publisher** (p. 443) 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 **DDS_Publisher** (p. 443) to which the **DDS_DataWriter** (p. 485) belongs and the entity name of of the **DDS_DataWriter** (p. 485) itself, separated by a double colon "::". For example: MyPublisherName::MyDataWriterName

The plain name contains the **DDS_DataWriter** (p. 485) name only. In this situation it is implied that the **DDS_DataWriter** (p. 485) belongs to the implicit **DDS_Publisher** (p. 443) so the use of a plain name is equivalent to specifying a fully qualified name with the **DDS_Publisher** (p. 443) name part being "implicit". For example: the plain name "MyDataWriterName" is equivalent to specifying the fully qualified name "implicit::MyDataWriterName"

The **DDS_DataWriter** (p. 485) is only looked up within the **DDS_Publisher** (p. 443) specified in the fully qualified name, or within the implicit **DDS_Publisher** (p. 443) if the name was not fully qualified.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>datawriter_full_name</i>	<< <i>in</i> >> (p. 1013) Entity name or fully-qualified entity name of the DDS_DataWriter (p. 485).

Returns

The first **DDS_DataWriter** (p. 485) found with the specified name or NULL if it is not found.

See also

DDS_Publisher_lookup_datawriter_by_name (p. 469)

DDS_DomainParticipant_lookup_publisher_by_name (p. 156)

5.4.4.105 DDS_DomainParticipant_lookup_datareader_by_name()

```
DDS_DataReader * DDS_DomainParticipant_lookup_datareader_by_name (
    DDS_DomainParticipant * self,
    const char * datareader_full_name )
```

<<*extension*>> (p. 1012) Retrieves up a **DDS_DataReader** (p. 615) by its entity name in this **DDS_DomainParticipant** (p. 77).

Every **DDS_DataReader** (p. 615) in the system has an entity name which is configured and stored in the EntityName policy, **ENTITY_NAME** (p. 1290).

Every **DDS_Subscriber** (p. 572) in the system has an entity name which is also configured and stored in the EntityName policy, **ENTITY_NAME** (p. 1290).

This operation retrieves a **DDS_DataReader** (p. 615) within a **DDS_Subscriber** (p. 572) given the specified name which encodes both to the **DDS_DataReader** (p. 615) and the **DDS_Subscriber** (p. 572) name.

If there are several **DDS_DataReader** (p. 615) with the same name within the corresponding **DDS_Subscriber** (p. 572) 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 **DDS_Subscriber** (p. 572) to which the **DDS_DataReader** (p. 615) belongs and the entity name of of the **DDS_DataReader** (p. 615) itself, separated by a double colon "::". For example: MySubscriberName::MyDataReaderName

The plain name contains the **DDS_DataReader** (p. 615) name only. In this situation it is implied that the **DDS_DataReader** (p. 615) belongs to the implicit **DDS_Subscriber** (p. 572) so the use of a plain name is equivalent to specifying a fully qualified name with the **DDS_Subscriber** (p. 572) name part being "implicit". For example: the plain name "MyDataReaderName" is equivalent to specifying the fully qualified name "implicit::MyDataReaderName"

The **DDS_DataReader** (p. 615) is only looked up within the **DDS_Subscriber** (p. 572) specified in the fully qualified name, or within the implicit **DDS_Subscriber** (p. 572) if the name was not fully qualified.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>datareader_full_name</i>	<< <i>in</i> >> (p. 1013) Full entity name of the DDS_DataReader (p. 615).

Returns

The first **DDS_DataReader** (p. 615) found with the specified name or NULL if it is not found.

See also

DDS_Subscriber_lookup_datareader_by_name (p. 598)

DDS_DomainParticipant_lookup_subscriber_by_name (p. 157)

5.4.5 Variable Documentation

5.4.5.1 DDS_DOMAINPARTICIPANTFACTORY_QOS_PRINT_ALL

```
const struct DDS_DomainParticipantFactoryQos* DDS_DOMAINPARTICIPANTFACTORY_QOS_PRINT_ALL [extern]
```

Special value which can be supplied to **DDS_DomainParticipantFactoryQos_to_string_w_params** (p. 35) indicating that all of the QoS should be printed.

The **DDS_DomainParticipantFactoryQos_to_string_w_params** (p. 35) API accepts a base QoS profile as one of its arguments. The resultant string only contains the differences with respect to the supplied base QoS profile. Supplying the **DDS_DOMAINPARTICIPANTFACTORY_QOS_PRINT_ALL** (p. 160) sentinel value as the base QoS will result in all of the QoS being printed.

Note that there are some QoS policies and fields which are not intended for public use. Even when **DDS_↵DOMAINPARTICIPANTFACTORY_QOS_PRINT_ALL** (p. 160) is supplied as the base, these will not be printed unless they differ from the documented default. If you want to see their values, you must use the `print_private` field within the **DDS_QosPrintFormat** (p. 1926) structure.

This value should only be used as the base parameter to the **DDS_DomainParticipantFactoryQos_to_string_w_↵params** (p. 35) API.

5.4.5.2 DDS_DOMAINPARTICIPANT_QOS_PRINT_ALL

```
const struct DDS_DomainParticipantQos* DDS_DOMAINPARTICIPANT_QOS_PRINT_ALL [extern]
```

Special value which can be supplied to **DDS_DomainParticipantQos_to_string_w_params** (p. 81) indicating that all of the QoS should be printed.

The **DDS_DomainParticipantQos_to_string_w_params** (p. 81) API accepts a base QoS profile as one of its arguments. The resultant string only contains the differences with respect to the supplied base QoS profile. Supplying the **DDS_DOMAINPARTICIPANT_QOS_PRINT_ALL** (p. 160) sentinel value as the base QoS will result in all of the QoS being printed.

Note that there are some QoS policies and fields which are not intended for public use. Even when **DDS_↵DOMAINPARTICIPANT_QOS_PRINT_ALL** (p. 160) is supplied as the base, these will not be printed unless they differ from the documented default. If you want to see their values, you must use the `print_private` field within the **DDS_Qos_↵PrintFormat** (p. 1926) structure.

This value should only be used as the base parameter to the **DDS_DomainParticipantQos_to_string_w_params** (p. 81) API.

5.4.5.3 DDS_TOPIC_QOS_DEFAULT

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

Special value for creating a **DDS_Topic** (p. 175) with default QoS.

When used in **DDS_DomainParticipant_create_topic** (p. 115), this special value is used to indicate that the **DDS_Topic** (p. 175) should be created with the default **DDS_Topic** (p. 175) QoS by means of the operation **get_default_topic_qos** and using the resulting QoS to create the **DDS_Topic** (p. 175).

When used in **DDS_DomainParticipant_set_default_topic_qos** (p. 87), 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 **DDS_DomainParticipant_set_default_topic_qos** (p. 87) operation had never been called.

When used in **DDS_Topic_set_qos** (p. 197), this special value is used to indicate that the QoS of the **DDS_Topic** (p. 175) should be changed to match the current default QoS set in the **DDS_DomainParticipant** (p. 77) that the **DDS_Topic** (p. 175) 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 **DDS_DomainParticipant_get_default_topic_qos** (p. 86).

See also

DDS_DomainParticipant_create_topic (p. 115)

DDS_DomainParticipant_set_default_topic_qos (p. 87)

DDS_Topic_set_qos (p. 197)

Examples

HelloWorld_publisher.c, and **HelloWorld_subscriber.c**.

5.4.5.4 DDS_PUBLISHER_QOS_DEFAULT

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

Special value for creating a **DDS_Publisher** (p. 443) with default QoS.

When used in **DDS_DomainParticipant_create_publisher** (p. 103), this special value is used to indicate that the **DDS_Publisher** (p. 443) should be created with the default **DDS_Publisher** (p. 443) QoS by means of the operation **get_default_publisher_qos** and using the resulting QoS to create the **DDS_Publisher** (p. 443).

When used in **DDS_DomainParticipant_set_default_publisher_qos** (p. 89), 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 **DDS_DomainParticipant_set_default_publisher_qos** (p. 89) operation had never been called.

When used in **DDS_Publisher_set_qos** (p. 462), this special value is used to indicate that the QoS of the **DDS_Publisher** (p. 443) should be changed to match the current default QoS set in the **DDS_DomainParticipant** (p. 77) that the **DDS_Publisher** (p. 443) 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 **DDS_DomainParticipant_get_default_publisher_qos** (p. 89).

See also

DDS_DomainParticipant_create_publisher (p. 103)

DDS_DomainParticipant_set_default_publisher_qos (p. 89)

DDS_Publisher_set_qos (p. 462)

Examples

HelloWorld_publisher.c.

5.4.5.5 DDS_SUBSCRIBER_QOS_DEFAULT

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

Special value for creating a **DDS_Subscriber** (p. 572) with default QoS.

When used in **DDS_DomainParticipant_create_subscriber** (p. 106), this special value is used to indicate that the **DDS_Subscriber** (p. 572) should be created with the default **DDS_Subscriber** (p. 572) QoS by means of the operation **get_default_subscriber_qos** and using the resulting QoS to create the **DDS_Subscriber** (p. 572).

When used in **DDS_DomainParticipant_set_default_subscriber_qos** (p. 94), 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 **DDS_DomainParticipant_set_default_subscriber_qos** (p. 94) operation had never been called.

When used in **DDS_Subscriber_set_qos** (p. 592), this special value is used to indicate that the QoS of the **DDS_Subscriber** (p. 572) should be changed to match the current default QoS set in the **DDS_DomainParticipant** (p. 77) that the **DDS_Subscriber** (p. 572) 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 **DDS_DomainParticipant_get_default_subscriber_qos** (p. 94).

See also

DDS_DomainParticipant_create_subscriber (p. 106)

DDS_DomainParticipant_get_default_subscriber_qos (p. 94)

DDS_Subscriber_set_qos (p. 592)

Examples

HelloWorld_subscriber.c.

5.4.5.6 DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT

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

<<**extension**>> (p. 1012) Special value for creating a **DDS_FlowController** (p. 559) with default property.

When used in **DDS_DomainParticipant_create_flowcontroller** (p. 124) and **DDS_FlowController_set_property** (p. 562), this special value represents the set of values specified on the last successful call to **DDS_DomainParticipant_set_default_flowcontroller_property** (p. 99), or else, if the call was never made, the default values listed in **DDS_FlowControllerProperty_t** (p. 1799).

When used in **DDS_DomainParticipant_set_default_flowcontroller_property** (p. 99), this special value indicates that the default QoS should be reset back to the initial value that would be used if the **DDS_DomainParticipant_set_default_flowcontroller_property** (p. 99) 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 **DDS_DomainParticipant_get_default_flowcontroller_property** (p. 98).

See also

DDS_DomainParticipant_create_flowcontroller (p. 124)

DDS_DomainParticipant_set_default_flowcontroller_property (p. 99)

DDS_FlowController_set_property (p. 562)

5.4.5.7 DDS_PUBLISHER_QOS_PRINT_ALL

```
const struct DDS_PublisherQos* DDS_PUBLISHER_QOS_PRINT_ALL [extern]
```

Special value which can be supplied to **DDS_PublisherQos_to_string_w_params** (p. 446) indicating that all of the QoS should be printed.

The **DDS_PublisherQos_to_string_w_params** (p. 446) API accepts a base QoS profile as one of its arguments. The resultant string only contains the differences with respect to the supplied base QoS profile. Supplying the **DDS_PUBLISHER_QOS_PRINT_ALL** (p. 163) sentinel value as the base QoS will result in all of the QoS being printed.

Note that there are some QoS policies and fields which are not intended for public use. Even when **DDS_PUBLISHER_QOS_PRINT_ALL** (p. 163) is supplied as the base, these will not be printed unless they differ from the documented default. If you want to see their values, you must use the `print_private` field within the **DDS_QosPrintFormat** (p. 1926) structure.

This value should only be used as the base parameter to the **DDS_PublisherQos_to_string_w_params** (p. 446) API.

5.4.5.8 DDS_SQLFILTER_NAME

```
const char* const DDS_SQLFILTER_NAME
```

<<**extension**>> (p. 1012) The name of the built-in SQL filter that can be used with ContentFilteredTopics and MultiChannel DataWriters.

See also

Queries and Filters Syntax (p. 738)

5.4.5.9 DDS_STRINGMATCHFILTER_NAME

```
const char* const DDS_STRINGMATCHFILTER_NAME
```

<<**extension**>> (p. 1012) 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. 738)

5.4.5.10 DDS_SUBSCRIBER_QOS_PRINT_ALL

```
const struct DDS_SubscriberQos* DDS_SUBSCRIBER_QOS_PRINT_ALL [extern]
```

Special value which can be supplied to **DDS_SubscriberQos_to_string_w_params** (p. 576) indicating that all of the QoS should be printed.

The **DDS_SubscriberQos_to_string_w_params** (p. 576) API accepts a base QoS profile as one of its arguments. The resultant string only contains the differences with respect to the supplied base QoS profile. Supplying the **DDS_SUBSCRIBER_QOS_PRINT_ALL** (p. 164) sentinel value as the base QoS will result in all of the QoS being printed.

Note that there are some QoS policies and fields which are not intended for public use. Even when **DDS_SUBSCRIBER_QOS_PRINT_ALL** (p. 164) is supplied as the base, these will not be printed unless they differ from the documented default. If you want to see their values, you must use the `print_private` field within the **DDS_QosPrintFormat** (p. 1926) structure.

This value should only be used as the base parameter to the **DDS_SubscriberQos_to_string_w_params** (p. 576) API.

5.4.5.11 DDS_TOPIC_QOS_PRINT_ALL

```
const struct DDS_TopicQos* DDS_TOPIC_QOS_PRINT_ALL [extern]
```

Special value which can be supplied to **DDS_TopicQos_to_string_w_params** (p. 190) indicating that all of the QoS should be printed.

The **DDS_TopicQos_to_string_w_params** (p. 190) API accepts a base QoS profile as one of its arguments. The resultant string only contains the differences with respect to the supplied base QoS profile. Supplying the **DDS_TOPIC_QOS_PRINT_ALL** (p. 164) sentinel value as the base QoS will result in all of the QoS being printed.

Note that there are some QoS policies and fields which are not intended for public use. Even when **DDS_TOPIC_QOS_PRINT_ALL** (p. 164) is supplied as the base, these will not be printed unless they differ from the documented default. If you want to see their values, you must use the `print_private` field within the **DDS_QosPrintFormat** (p. 1926) structure.

This value should only be used as the base parameter to the **DDS_TopicQos_to_string_w_params** (p. 190) API.

5.5 Built-in Topics

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

Modules

- **Participant Built-in Topics**

Builtin topic for accessing information about the DomainParticipants discovered by RTI Connex.

- **Topic Built-in Topics**

- **Publication Built-in Topics**

Builtin topic for accessing information about the Publications discovered by RTI Connex.

- **Subscription Built-in Topics**

Builtin topic for accessing information about the Subscriptions discovered by RTI Connex.

- **ServiceRequest Built-in Topic**

Builtin topic for accessing requests from different services within RTI Connex.

- **Common types and functions**

Types and functions related to the built-in topics.

5.5.1 Detailed Description

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

RTI Connex 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 **DDS_DataReader** (p. 615) 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 **DDS_Listener** (p. 1816) or the **DDS_Condition** (p. 1370) mechanisms.

The built-in data-readers all belong to a built-in **DDS_Subscriber** (p. 572), which can be retrieved by using the function **DDS_DomainParticipant_get_builtin_subscriber** (p. 129). The built-in **DDS_DataReader** (p. 615) objects can be retrieved by using the operation **DDS_Subscriber_lookup_datareader** (p. 586), with the topic name as a parameter.

Built-in entities have default listener settings as well. The built-in **DDS_Subscriber** (p. 572) 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 **DDS_DataReader** (p. 615) will not provide data pertaining to entities created from the same **DDS_DomainParticipant** (p. 77) 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 DomainParticipant was created in a disabled state (see **DDS_EntityFactoryQosPolicy** (p. 1788)), the built-in subscriber and readers will need to be enabled explicitly (by the **DDS_Entity_enable** (p. 1363) operation).

Refer to **DDS_ParticipantBuiltinTopicData** (p. 1870), **DDS_TopicBuiltinTopicData** (p. 2034), **DDS_SubscriptionBuiltinTopicData** (p. 2012) and **DDS_PublicationBuiltinTopicData** (p. 1905) for a description of all the built-in topics and their contents.

The QoS of the built-in **DDS_Subscriber** (p. 572) and **DDS_DataReader** (p. 615) objects is given by the following table:

Table 5.150 QoS of built-in DDS_Subscriber (p. 572) and DDS_DataReader (p. 615)

QoS	Value
DDS_UserDataQosPolicy (p. 2086)	0-length sequence
DDS_TopicDataQosPolicy (p. 2040)	0-length sequence
DDS_GroupDataQosPolicy (p. 1803)	0-length sequence
DDS_DurabilityQosPolicy (p. 1759)	DDS_TRANSIENT_LOCAL_DURABILITY_QOS (p. 1288)
DDS_DurabilityServiceQosPolicy (p. 1764)	Does not apply as DDS_DurabilityQosPolicyKind (p. 1287) is DDS_TRANSIENT_LOCAL_DURABILITY_QOS (p. 1288)
DDS_PresentationQosPolicy (p. 1890)	access_scope = DDS_TOPIC_PRESENTATION_QOS (p. 1305) coherent_access = DDS_BOOLEAN_FALSE (p. 1201) ordered_access = DDS_BOOLEAN_FALSE (p. 1201)

QoS	Value
DDS_DeadlineQosPolicy (p. 1684)	Period = infinite
DDS_LatencyBudgetQosPolicy (p. 1813)	duration = 0
DDS_OwnershipQosPolicy (p. 1863)	DDS_SHARED_OWNERSHIP_QOS (p. 1302)
DDS_OwnershipStrengthQosPolicy (p. 1868)	value = 0
DDS_LivelinessQosPolicy (p. 1823)	kind = DDS_AUTOMATIC_LIVELINESS_QOS (p. 1297) lease_duration = 0
DDS_TimeBasedFilterQosPolicy (p. 2032)	minimum_separation = 0
DDS_PartitionQosPolicy (p. 1881)	0-length sequence
DDS_ReliabilityQosPolicy (p. 1936)	kind = DDS_RELIABLE_RELIABILITY_QOS (p. 1323) max_blocking_time = 100 milliseconds
DDS_DestinationOrderQosPolicy (p. 1686)	DDS_BY_RECEPTION_TIMESTAMP ↔ DESTINATIONORDER_QOS (p. 1274)
DDS_HistoryQosPolicy (p. 1805)	kind = DDS_KEEP_LAST_HISTORY_QOS (p. 1293) depth = 1
DDS_ResourceLimitsQosPolicy (p. 1949)	max_samples = DDS_LENGTH_UNLIMITED (p. 1326) max_instances = DDS_LENGTH_UNLIMITED (p. 1326) max_samples_per_instance = DDS_LENGTH_UNLIMITED (p. 1326)
DDS_ReaderDataLifecycleQosPolicy (p. 1930)	autopurge_nowriter_samples_delay = infinite autopurge_disposed_samples_delay = infinite
DDS_EntityFactoryQosPolicy (p. 1788)	autoenable_created_entities = DDS_BOOLEAN_TRUE (p. 1201)

5.6 Topic Module

Contains the **DDS_Topic** (p. 175), **DDS_ContentFilteredTopic** (p. 177), and **DDS_MultiTopic** (p. 185) classes, the **DDS_TopicListener** (p. 2041) interface, and more generally, all that is needed by an application to define **DDS_Topic** (p. 175) objects and attach QoS policies to them.

Modules

- **Topics**
DDS_Topic (p. 175) entity and associated elements
- **FlatData Topic-Types**
<<extension>> (p. 1012) FlatData Language Binding for IDL topic-types
- **Zero Copy Transfer Over Shared Memory**
<<extension>> (p. 1012) 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. 1012) A **DDS_TypeCode** (p. 2071) 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. 313) 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. 1870), **DDS_PublicationBuiltinTopicData** (p. 1905), **DDS_SubscriptionBuiltinTopicData** (p. 2012) to describe Trust Plugins configuration.*

- **Dynamic Data**

<<*extension*>> (p. 1012) 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.

5.6.1 Detailed Description

Contains the **DDS_Topic** (p. 175), **DDS_ContentFilteredTopic** (p. 177), and **DDS_MultiTopic** (p. 185) classes, the **DDS_TopicListener** (p. 2041) interface, and more generally, all that is needed by an application to define **DDS_Topic** (p. 175) objects and attach QoS policies to them.

5.7 Topics

DDS_Topic (p. 175) entity and associated elements

Data Structures

- 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. 1228)

- struct **DDS_TopicQos**

*QoS policies supported by a **DDS_Topic** (p. 175) entity.*

- struct **DDS_TopicListener**

<<*interface*>> (p. 1013) **DDS_Listener** (p. 1816) for **DDS_Topic** (p. 175) entities.

- struct **DDS_ContentFilter**

<<*interface*>> (p. 1013) Interface to be used by a custom filter of a **DDS_ContentFilteredTopic** (p. 177)

- struct **DDS_ExpressionProperty**

*Provides additional information about the filter expression passed to **DDS_ContentFilter::writer_compile** (p. 1573).*

- struct **DDS_FilterSampleInfo**

Provides meta information associated with the sample.

Macros

- **#define DDS_PrintFormatProperty_INITIALIZER**
*Static initializer for **DDS_PrintFormatProperty** (p. 1894) objects.*
- **#define DDS_InconsistentTopicStatus_INITIALIZER**
Initializer for new status instances.
- **#define DDS_TopicQos_INITIALIZER**
Initializer for new QoS instances.
- **#define DDS_TopicListener_INITIALIZER**
*Initializer for new **DDS_TopicListener** (p. 2041).*
- **#define DDS_ContentFilter_INITIALIZER**
*Initializer for new **DDS_ContentFilter** (p. 1571).*

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.
- **typedef struct DDS_TopicDescriptionImpl DDS_TopicDescription**
*<<interface>> (p. 1013) Base class for **DDS_Topic** (p. 175), **DDS_ContentFilteredTopic** (p. 177), and **DDS_MultiTopic** (p. 185).*
- **typedef struct DDS_TopicWrapperI DDS_Topic**
<<interface>> (p. 1013) The most basic description of the data to be published and subscribed.
- **typedef void(* DDS_TopicListener_InconsistentTopicCallback)** (void *listener_data, **DDS_Topic** *topic, const struct **DDS_InconsistentTopicStatus** *status)
*Prototype of **DDS_TopicListener::on_inconsistent_topic** (p. 2042).*
- **typedef struct DDS_ContentFilteredTopicWrapperI DDS_ContentFilteredTopic**
*<<interface>> (p. 1013) Specialization of **DDS_TopicDescription** (p. 175) that allows for content-based subscriptions.*
- **typedef DDS_ReturnCode_t(* DDS_ContentFilterCompileFunction)** (void *filter_data, void **new_compile_data, const char *expression, const struct **DDS_StringSeq** *parameters, const struct **DDS_TypeCode** *type_code, const char *type_class_name, void *old_compile_data)
*Prototype of **DDS_ContentFilter::compile** (p. 1572).*
- **typedef DDS_Boolean(* DDS_ContentFilterEvaluateFunction)** (void *filter_data, void *compile_data, const void *sample, const struct **DDS_FilterSampleInfo** *meta_data)
*Prototype of **DDS_ContentFilter::evaluate** (p. 1574).*
- **typedef void(* DDS_ContentFilterFinalizeFunction)** (void *filter_data, void *compile_data)
*Prototype of **DDS_ContentFilter::finalize** (p. 1576).*
- **typedef void(* DDS_ContentFilterWriterFinalizeFunction)** (void *filter_data, void *writer_filter_data, const struct **DDS_Cookie_t** *cookie)
*Prototype of **DDS_ContentFilter::writer_finalize** (p. 1577).*
- **typedef DDS_ReturnCode_t(* DDS_ContentFilterWriterAttachFunction)** (void *filter_data, void **writer_filter_data, void *reserved)
*Prototype of **DDS_ContentFilter::writer_attach** (p. 1577).*
- **typedef void(* DDS_ContentFilterWriterDetachFunction)** (void *filter_data, void *writer_filter_data)
*Prototype of **DDS_ContentFilter::writer_detach** (p. 1578).*
- **typedef DDS_ReturnCode_t(* DDS_ContentFilterWriterCompileFunction)** (void *filter_data, void *writer_filter_data, struct **DDS_ExpressionProperty** *prop, const char *expression, const struct **DDS_StringSeq** *parameters, const struct **DDS_TypeCode** *type_code, const char *type_class_name, const struct **DDS_Cookie_t** *cookie)

Prototype of **DDS_ContentFilter::writer_compile** (p. 1573).

- typedef struct **DDS_CookieSeq** **(***DDS_ContentFilterWriterEvaluateFunction***)* (void *filter_data, void *writer_filter_data, const void *sample, const struct **DDS_FilterSampleInfo** *meta_data)

Prototype of **DDS_ContentFilter::writer_evaluate** (p. 1576).

- typedef void(*****DDS_ContentFilterWriterReturnLoanFunction***)* (void *filter_data, void *writer_filter_data, struct **DDS_CookieSeq** *cookies)

Prototype of **DDS_ContentFilter::writer_return_loan** (p. 1578).

- typedef struct DDS_MultiTopicImpl **DDS_MultiTopic**

[Not supported (optional)] <<interface>> (p. 1013) A specialization of **DDS_TopicDescription** (p. 175) that allows subscriptions that combine/filter/rearrange data coming from several topics.

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_ReturnCode_t** **DDS_InconsistentTopicStatus_initialize** (struct **DDS_InconsistentTopicStatus** *self)
Initializer for new status instances.
- **DDS_ReturnCode_t** **DDS_InconsistentTopicStatus_copy** (struct **DDS_InconsistentTopicStatus** *self, const struct **DDS_InconsistentTopicStatus** *source)
Copy the contents of the given status into this status.
- **DDS_ReturnCode_t** **DDS_InconsistentTopicStatus_finalize** (struct **DDS_InconsistentTopicStatus** *self)
Free any dynamic memory allocated by status instances.
- **DDS_Boolean** **DDS_InconsistentTopicStatus_equals** (const struct **DDS_InconsistentTopicStatus** *left, const struct **DDS_InconsistentTopicStatus** *right)
*Compares two **DDS_InconsistentTopicStatus** (p. 1808) for equality.*
- **DDS_Boolean** **DDS_TopicQos_equals** (const struct **DDS_TopicQos** *self, const struct **DDS_TopicQos** *other)
*Compares two **DDS_TopicQos** (p. 2042) for equality.*
- **DDS_ReturnCode_t** **DDS_TopicQos_print** (const struct **DDS_TopicQos** *self)
*Prints this **DDS_TopicQos** (p. 2042) to stdout.*
- **DDS_ReturnCode_t** **DDS_TopicQos_to_string** (const struct **DDS_TopicQos** *self, char *string, **DDS_↔** **UnsignedLong** *string_size)
*Obtains a string representation of this **DDS_TopicQos** (p. 2042).*
- **DDS_ReturnCode_t** **DDS_TopicQos_to_string_w_params** (const struct **DDS_TopicQos** *self, char *string, **DDS_UnsignedLong** *string_size, const struct **DDS_TopicQos** *base, const struct **DDS_QosPrintFormat** *format)
*Obtains a string representation of this **DDS_TopicQos** (p. 2042).*
- **DDS_ReturnCode_t** **DDS_TopicQos_initialize** (struct **DDS_TopicQos** *self)
Initializer for new QoS instances.
- **DDS_ReturnCode_t** **DDS_TopicQos_finalize** (struct **DDS_TopicQos** *self)
*Free any dynamic memory allocated by the policies in this **DDS_TopicQos** (p. 2042).*

- **DDS_ReturnCode_t DDS_TopicQos_copy** (struct **DDS_TopicQos** *self, const struct **DDS_TopicQos** *source)
Copy the contents of the given QoS into this QoS.
- const char * **DDS_TopicDescription_get_type_name** (**DDS_TopicDescription** *self)
Get the associated `type_name`.
- const char * **DDS_TopicDescription_get_name** (**DDS_TopicDescription** *self)
*Get the name used to create this **DDS_TopicDescription** (p. 175) .*
- **DDS_DomainParticipant * DDS_TopicDescription_get_participant** (**DDS_TopicDescription** *self)
*Get the **DDS_DomainParticipant** (p. 77) to which the **DDS_TopicDescription** (p. 175) belongs.*
- **DDS_Entity * DDS_Topic_as_entity** (**DDS_Topic** *topic)
*Access a **DDS_Topic** (p. 175)'s **DDS_Entity** (p. 1360) supertype instance.*
- **DDS_TopicDescription * DDS_Topic_as_topicdescription** (**DDS_Topic** *topic)
*Access a **DDS_Topic** (p. 175)'s **DDS_TopicDescription** (p. 175) supertype instance.*
- **DDS_Topic * DDS_Topic_narrow** (**DDS_TopicDescription** *self)
*Narrow the given **DDS_TopicDescription** (p. 175) pointer to a **DDS_Topic** (p. 175) pointer.*
- **DDS_Topic * DDS_Topic_narrow_from_entity** (**DDS_Entity** *self)
*Narrow the given **DDS_Entity** (p. 1360) pointer to a **DDS_Topic** (p. 175) pointer.*
- **DDS_ReturnCode_t DDS_Topic_get_inconsistent_topic_status** (**DDS_Topic** *self, struct **DDS_InconsistentTopicStatus** *status)
*Allows the application to retrieve the **DDS_INCONSISTENT_TOPIC_STATUS** (p. 1228) status of a **DDS_Topic** (p. 175).*
- **DDS_ReturnCode_t DDS_Topic_set_qos** (**DDS_Topic** *self, const struct **DDS_TopicQos** *qos)
Set the topic QoS.
- **DDS_ReturnCode_t DDS_Topic_set_qos_with_profile** (**DDS_Topic** *self, const char *library_name, const char *profile_name)
<<extension>> (p. 1012) Change the QoS of this topic using the input XML QoS profile.
- **DDS_ReturnCode_t DDS_Topic_get_qos** (**DDS_Topic** *self, struct **DDS_TopicQos** *qos)
Get the topic QoS.
- **DDS_ReturnCode_t DDS_Topic_set_listener** (**DDS_Topic** *self, const struct **DDS_TopicListener** *l, **DDS_StatusMask** mask)
Set the topic listener.
- struct **DDS_TopicListener** **DDS_Topic_get_listener** (**DDS_Topic** *self)
Get the topic listener.
- **DDS_ReturnCode_t DDS_Topic_get_listenerX** (**DDS_Topic** *self, struct **DDS_TopicListener** *listener)
<<extension>> (p. 1012) Get the topic listener.
- **DDS_TopicDescription * DDS_ContentFilteredTopic_as_topicdescription** (**DDS_ContentFilteredTopic** *contentFilteredTopic)
*Access a **DDS_ContentFilteredTopic** (p. 177)'s supertype instance.*
- **DDS_ContentFilteredTopic * DDS_ContentFilteredTopic_narrow** (**DDS_TopicDescription** *self)
*Narrow the given **DDS_TopicDescription** (p. 175) pointer to a **DDS_ContentFilteredTopic** (p. 177) pointer.*
- const char * **DDS_ContentFilteredTopic_get_filter_expression** (**DDS_ContentFilteredTopic** *self)
Get the `filter_expression`.
- **DDS_ReturnCode_t DDS_ContentFilteredTopic_get_expression_parameters** (**DDS_ContentFilteredTopic** *self, struct **DDS_StringSeq** *parameters)
Get the `expression_parameters`.
- **DDS_ReturnCode_t DDS_ContentFilteredTopic_set_expression_parameters** (**DDS_ContentFilteredTopic** *self, const struct **DDS_StringSeq** *parameters)
Set the `expression_parameters`.

- **DDS_ReturnCode_t DDS_ContentFilteredTopic_set_expression** (DDS_ContentFilteredTopic *self, const char *expression, const struct DDS_StringSeq *parameters)
Set the filter_expression and expression_parameters.
- **DDS_ReturnCode_t DDS_ContentFilteredTopic_append_to_expression_parameter** (DDS_ContentFilteredTopic *self, const DDS_Long index, const char *val)
<<extension>> (p. 1012) Appends a string term to the specified parameter string.
- **DDS_ReturnCode_t DDS_ContentFilteredTopic_remove_from_expression_parameter** (DDS_ContentFilteredTopic *self, const DDS_Long index, const char *val)
<<extension>> (p. 1012) Removes a string term from the specified parameter string.
- **DDS_Topic * DDS_ContentFilteredTopic_get_related_topic** (DDS_ContentFilteredTopic *self)
Get the related_topic.
- **DDS_TopicDescription * DDS_MultiTopic_as_topicdescription** (DDS_MultiTopic *multiTopic)
Access a DDS_MultiTopic (p. 185)'s supertype instance.
- **DDS_MultiTopic * DDS_MultiTopic_narrow** (DDS_TopicDescription *self)
Narrow the given DDS_TopicDescription (p. 175) pointer to a DDS_MultiTopic (p. 185) pointer.
- **const char * DDS_MultiTopic_get_subscription_expression** (DDS_MultiTopic *self)
Get the expression for this DDS_MultiTopic (p. 185).
- **DDS_ReturnCode_t DDS_MultiTopic_get_expression_parameters** (DDS_MultiTopic *self, struct DDS_StringSeq *parameters)
Get the expression parameters.
- **DDS_ReturnCode_t DDS_MultiTopic_set_expression_parameters** (DDS_MultiTopic *self, const struct DDS_StringSeq *parameters)
Set the expression_parameters.

Variables

- **const struct DDS_PrintFormatProperty DDS_PRINT_FORMAT_PROPERTY_DEFAULT**
Sentinel constant indicating default values for DDS_PrintFormatProperty (p. 1894).
- **const struct DDS_FilterSampleInfo DDS_FILTERSAMPLEINFO_DEFAULT**
Initializer for new DDS_FilterSampleInfo (p. 1796).

5.7.1 Detailed Description

DDS_Topic (p. 175) entity and associated elements

5.7.2 Macro Definition Documentation

5.7.2.1 DDS_PrintFormatProperty_INITIALIZER

```
#define DDS_PrintFormatProperty_INITIALIZER
```

Static initializer for **DDS_PrintFormatProperty** (p. 1894) objects.

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

```
struct DDS_PrintFormatProperty props = DDS_PrintFormatProperty_INITIALIZER;
```

See also

DDS_PrintFormatProperty (p. 1894)

5.7.2.2 DDS_InconsistentTopicStatus_INITIALIZER

```
#define DDS_InconsistentTopicStatus_INITIALIZER
```

Initializer for new status instances.

New **DDS_InconsistentTopicStatus** (p. 1808) instances that are stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those fields that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a status structure is to initialize it on the stack at the time of its creation. **DDS_InconsistentTopicStatus_finalize** (p. 188) should be called to free the contained fields that use dynamic memory:

```
struct DDS_InconsistentTopicStatus myStatus = DDS_InconsistentTopicStatus_INITIALIZER;
```

```
DDS_Topic_get_inconsistent_topic(myTopic, &myStatus);
```

```
DDS_InconsistentTopicStatus_finalize(&myStatus);
```

See also

DDS_InconsistentTopicStatus_initialize (p. 186)

DDS_Topic_get_inconsistent_topic_status (p. 196)

DDS_InconsistentTopicStatus_finalize (p. 188)

5.7.2.3 DDS_TopicQos_INITIALIZER

```
#define DDS_TopicQos_INITIALIZER
```

Initializer for new QoS instances.

New **DDS_TopicQos** (p. 2042) instances stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those contained QoS policies that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a new QoS structure is to initialize it on the stack at the time of its creation:

```
struct DDS_TopicQos myQos = DDS_TopicQos_INITIALIZER;
```

Note that the above assignment is not a substitute for calling **DDS_DomainParticipant_get_default_topic_qos** (p. 86) or **DDS_Topic_get_qos** (p. 199); one of those should be called subsequently to setting the QoS of a new or existing entity. **DDS_TopicQos_finalize** (p. 192) should be called to free the contained QoS policies that use dynamic memory:

```
struct DDS_TopicQos myQos = DDS_TopicQos_INITIALIZER;
```

```
DDS_DomainParticipant_get_default_topic_qos(myParticipant, &myQos);
```

```
DDS_Topic_set_qos(myTopic, &myQos);
```

```
DDS_TopicQos_finalize(&myQos);
```

See also

DDS_DomainParticipant_get_default_topic_qos (p. 86)

DDS_TopicQos_finalize (p. 192)

5.7.2.4 DDS_TopicListener_INITIALIZER

```
#define DDS_TopicListener_INITIALIZER
```

Initializer for new **DDS_TopicListener** (p. 2041).

All the new **DDS_TopicListener** (p. 2041) instances allocated in the stack should be initialized to this value. No memory is allocated.

```
struct DDS_TopicListener listener = DDS_TopicListener_INITIALIZER;
```

```
/* initialize listener functions */
```

```
listener.on_inconsistent_topic = ...;
```

```
DDS_Topic_set_listener(myTopic, &listener, mask);
```

See also

DDS_Topic_set_listener (p. 199)

DDS_TopicListener (p. 2041)

5.7.2.5 DDS_ContentFilter_INITIALIZER

```
#define DDS_ContentFilter_INITIALIZER
```

Initializer for new **DDS_ContentFilter** (p. 1571).

All the new **DDS_ContentFilter** (p. 1571) instances allocated in the stack should be initialized to this value. No memory is allocated.

See also

DDS_ContentFilter (p. 1571)

DDS_DomainParticipant_create_contentfilteredtopic_with_filter (p. 119)

5.7.3 Typedef Documentation

5.7.3.1 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.

To ensure that new objects are initialized to known values, assign them with the static initializer **DDS_PrintFormatProperty_INITIALIZER** (p. 172).

See also

DDS_PrintFormatProperty_INITIALIZER (p. 172)

5.7.3.2 DDS_TopicDescription

```
typedef struct DDS_TopicDescriptionImpl DDS_TopicDescription
```

<<interface>> (p. 1013) Base class for **DDS_Topic** (p. 175), **DDS_ContentFilteredTopic** (p. 177), and **DDS_MultiTopic** (p. 185).

DDS_TopicDescription (p. 175) 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 **DDS_TypeSupport** (p. 214).

DDS_TopicDescription (p. 175) has also a `name` that allows it to be retrieved locally.

See also

DDS_TypeSupport (p. 214), **FooTypeSupport** (p. 2114)

5.7.3.3 DDS_Topic

```
typedef struct DDS_TopicWrapperI DDS_Topic
```

<<*interface*>> (p. 1013) The most basic description of the data to be published and subscribed.

QoS:

DDS_TopicQos (p. 2042)

Status:

DDS_INCONSISTENT_TOPIC_STATUS (p. 1228), **DDS_InconsistentTopicStatus** (p. 1808)

Listener:

DDS_TopicListener (p. 2041)

A **DDS_Topic** (p. 175) is identified by its name, which must be unique in the whole domain. In addition (by virtue of extending **DDS_TopicDescription** (p. 175)) it fully specifies the type of the data that can be communicated when publishing or subscribing to the **DDS_Topic** (p. 175).

DDS_Topic (p. 175) is the only **DDS_TopicDescription** (p. 175) that can be used for publications and therefore associated with a **DDS_DataWriter** (p. 485).

The following operations may be called even if the **DDS_Topic** (p. 175) is not enabled. Other operations will fail with the value **DDS_RETCODE_NOT_ENABLED** (p. 1222) if called on a disabled **DDS_Topic** (p. 175):

- **DDS_Entity_enable** (p. 1363)
- **DDS_Topic_set_qos** (p. 197), **DDS_Topic_get_qos** (p. 199)
- **DDS_Topic_set_qos_with_profile** (p. 198)
- **DDS_Topic_set_listener** (p. 199), **DDS_Topic_get_listener** (p. 200)
- **DDS_Entity_get_statuscondition** (p. 1365), **DDS_Entity_get_status_changes** (p. 1365)
- **DDS_Topic_get_inconsistent_topic_status** (p. 196)

See also

Operations Allowed in Listener Callbacks (p. 1818)

5.7.3.4 DDS_TopicListener_InconsistentTopicCallback

```
typedef void(* DDS_TopicListener_InconsistentTopicCallback) (void *listener_data, DDS_Topic *topic,  
const struct DDS_InconsistentTopicStatus *status)
```

Prototype of **DDS_TopicListener::on_inconsistent_topic** (p. 2042).

Parameters

<i>listener_data</i>	<< out >> (p. 1013) Data associated with the listener when the listener is set
<i>topic</i>	<< out >> (p. 1013) Locally created DDS_Topic (p. 175) that triggers the listener callback
<i>status</i>	<< out >> (p. 1013) Current inconsistent status of the locally created DDS_Topic (p. 175)

5.7.3.5 DDS_ContentFilteredTopic

```
typedef struct DDS_ContentFilteredTopicWrapperI DDS_ContentFilteredTopic
```

<<**interface**>> (p. 1013) Specialization of **DDS_TopicDescription** (p. 175) that allows for content-based subscriptions.

It describes a more sophisticated subscription that indicates a **DDS_DataReader** (p. 615) does not want to necessarily see all values of each instance published under the **DDS_Topic** (p. 175). 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. 738) describes the syntax of `filter_expression` and `expression_←` parameters.

5.7.3.6 DDS_ContentFilterCompileFunction

```
typedef DDS_ReturnCode_t(* DDS_ContentFilterCompileFunction) (void *filter_data, void **new_←  
compile_data, const char *expression, const struct DDS_StringSeq *parameters, const struct DDS_←  
_TypeCode *type_code, const char *type_class_name, void *old_compile_data)
```

Prototype of **DDS_ContentFilter::compile** (p. 1572).

Compile an instance of the content filter according to the filter expression and parameters of the given data type.

This function 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 **DDS_ContentFilteredTopic** (p. 177) with the matching filter name is created, or when a **DDS_DataReader** (p. 615) 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

<i>filter_data</i>	<< <i>in</i> >> (p. 1013) The opaque pointer that the content filter was registered with in DDS_DomainParticipant_register_contentfilter (p. 120). Can be NULL. When a custom filter is registered, it is registered with a filter_data . This filter_data is user defined, e.g., it could be a reference to a program context. This is useful if the content filter functions are registered with different filter names. For instance, each filter name could have its own context and the DDS_ContentFilter::compile (p. 1572) function can access this context.

Parameters

<i>new_compile_data</i>	<< <i>out</i> >> (p. 1013) User specified opaque pointer of this instance of the content filter. This value is then passed to the DDS_ContentFilter::evaluate (p. 1574) and DDS_ContentFilter::finalize (p. 1576) functions for this instance of the content filter. Can be set to NULL .
<i>expression</i>	<< <i>in</i> >> (p. 1013) An ASCIIZ string with the filter expression. The memory used by this string is owned by RTI Connex and must not be freed. If you want to manipulate this string, you must first make a copy of it.
<i>parameters</i>	<< <i>in</i> >> (p. 1013) A string sequence with the expression parameters the DDS_ContentFilteredTopic (p. 177) was created with. The string sequence is equal (but not identical) to the string sequence passed to DDS_DomainParticipant_create_contentfilteredtopic (p. 118). Note that the sequence passed to the compile function is owned by RTI Connex and must not be referenced outside the compile function.
<i>type_code</i>	<< <i>in</i> >> (p. 1013) A pointer to the type code for the related DDS_Topic (p. 175) of the DDS_ContentFilteredTopic (p. 177). A type_code is a description of a type in terms of which types it contains (such as long, 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.
<i>type_class_name</i>	<< <i>in</i> >> (p. 1013) Fully qualified class name of the related DDS_Topic (p. 175).
<i>old_compile_data</i>	<< <i>in</i> >> (p. 1013) 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 DDS_ContentFilteredTopic (p. 177), 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.

Returns

One of the **Standard Return Codes** (p. 1221)

5.7.3.7 DDS_ContentFilterEvaluateFunction

```
typedef DDS_Boolean(* DDS_ContentFilterEvaluateFunction) (void *filter_data, void *compile_data,
const void *sample, const struct DDS_FilterSampleInfo *meta_data)
```

Prototype of **DDS_ContentFilter::evaluate** (p. 1574).

Evaluate whether the sample is passing the filter or not according to the sample content.

This function is called when a sample for a locally created **DDS_DataReader** (p. 615) associated with the filter is received, or when a sample for a discovered **DDS_DataReader** (p. 615) associated 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

<i>filter_data</i>	<< <i>in</i> >> (p. 1013) The opaque pointer that the content filter was registered with in DDS_DomainParticipant_register_contentfilter (p. 120). Can be NULL.

Parameters

<i>compile_data</i>	<< <i>in</i> >> (p. 1013) The last return value of the DDS_ContentFilter::compile (p. 1572) function for this instance of the content filter. Can be NULL .
<i>sample</i>	<< <i>in</i> >> (p. 1013) Pointer to a deserialized sample to be filtered
<i>meta_data</i>	<< <i>in</i> >> (p. 1013) Pointer to meta data associated with the sample.

Returns

The function must return 0 if the sample should be filtered out, non zero otherwise

5.7.3.8 DDS_ContentFilterFinalizeFunction

```
typedef void(* DDS_ContentFilterFinalizeFunction) (void *filter_data, void *compile_data)
```

Prototype of **DDS_ContentFilter::finalize** (p. 1576).

A previously compiled instance of the content filter is no longer in use and resources can now be cleaned up.

This function 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 **DDS_ContentFilteredTopic** (p. 177) with the matching filter name is deleted, or when a **DDS_DataReader** (p. 615) with a matching filter name is removed due to discovery.

This function is also called on all instances of the discovered **DDS_DataReader** (p. 615) with a matching filter name if the filter is unregistered with **DDS_DomainParticipant_unregister_contentfilter** (p. 121).

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

<i>filter_data</i>	<< <i>in</i> >> (p. 1013) The opaque pointer that the content filter was registered with in DDS_DomainParticipant_register_contentfilter (p. 120). Can be NULL.

Parameters

<i>compile_data</i>	<< <i>in</i> >> (p. 1013) The last return value of the DDS_ContentFilter::compile (p. 1572) function for this instance of the content filter. Can be NULL

5.7.3.9 DDS_ContentFilterWriterFinalizeFunction

```
typedef void(* DDS_ContentFilterWriterFinalizeFunction) (void *filter_data, void *writer_filter_↵
data, const struct DDS_Cookie_t *cookie)
```

Prototype of **DDS_ContentFilter::writer_finalize** (p. 1577).

A writer-side filtering API to clean up a previously compiled instance of the content filter.

This function is called to notify the filter implementation that the **DDS_DataWriter** (p. 485) is no longer matching with a **DDS_DataReader** (p. 615) for which it was previously performing writer-side filtering. This will allow the filter to purge any state it was maintaining for the **DDS_DataReader** (p. 615).

It is possible for multiple threads to be calling into this function at the same time.

Parameters

<i>filter_data</i>	<< <i>in</i> >> (p. 1013) The opaque pointer that the content filter was registered with in DDS_DomainParticipant_register_contentfilter (p. 120). Can be NULL.

Parameters

<i>writer_filter_data</i>	<< <i>in</i> >> (p. 1013) A pointer to the state created using DDS_ContentFilter::writer_attach (p. 1577).
<i>cookie</i>	DDS_Cookie_t (p. 1580) to uniquely identify DDS_DataReader (p. 615) for which DDS_ContentFilter::writer_finalize (p. 1577) was called.

5.7.3.10 DDS_ContentFilterWriterAttachFunction

```
typedef DDS_ReturnCode_t(* DDS_ContentFilterWriterAttachFunction) (void *filter_data, void **writer_↵
_filter_data, void *reserved)
```

Prototype of **DDS_ContentFilter::writer_attach** (p. 1577).

A writer-side filtering API to create some state that can facilitate filtering on the writer side.

This function is called to create some state required to perform filtering on the writer side using writer-side filtering APIs. This function will be called for every **DDS_DataWriter** (p. 485); it will be called only the first time the **DDS_DataWriter** (p. 485) matches a **DDS_DataReader** (p. 615) 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

<i>filter_data</i>	<< <i>in</i> >> (p. 1013) The opaque pointer that the content filter was registered with in DDS_DomainParticipant_register_contentfilter (p. 120). Can be NULL.

Parameters

<i>writer_filter_data</i>	<< <i>out</i> >> (p. 1013) A user-specified opaque pointer to some state created on the DDS_DataWriter (p. 485) that will help perform writer-side filtering efficiently.

Parameters

<i>reserved</i>	Reserved.

5.7.3.11 DDS_ContentFilterWriterDetachFunction

```
typedef void(* DDS_ContentFilterWriterDetachFunction) (void *filter_data, void *writer_filter_data)
```

Prototype of **DDS_ContentFilter::writer_detach** (p. 1578).

A writer-side filtering API to clean up a previously created state using **DDS_ContentFilter::writer_attach** (p. 1577).

This function is called to delete any state created using the **DDS_ContentFilter::writer_attach** (p. 1577) function. This function will be called when the **DDS_DataWriter** (p. 485) is deleted.

Parameters

<i>filter_data</i>	<< <i>in</i> >> (p. 1013) The opaque pointer that the content filter was registered in DDS_DomainParticipant_register_contentfilter (p. 120). Can be NULL.

Parameters

<i>writer_filter_data</i>	<< <i>in</i> >> (p. 1013) A pointer to the state created using DDS_ContentFilter::writer_attach (p. 1577).

5.7.3.12 DDS_ContentFilterWriterCompileFunction

```
typedef DDS_ReturnCode_t (* DDS_ContentFilterWriterCompileFunction) (void *filter_data, void *writer_data, struct DDS_ExpressionProperty *prop, const char *expression, const struct DDS_StringSeq *parameters, const struct DDS_TypeCode *type_code, const char *type_class_name, const struct DDS_Cookie_t *cookie)
```

Prototype of **DDS_ContentFilter::writer_compile** (p. 1573).

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching **DDS_DataReader** (p. 615).

This function is called when the **DDS_DataWriter** (p. 485) discovers a **DDS_DataReader** (p. 615) with a **DDS_ContentFilteredTopic** (p. 177) or when a **DDS_DataWriter** (p. 485) 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

<i>filter_data</i>	<< <i>in</i> >> (p. 1013) The opaque pointer that the content filter was registered with in DDS_DomainParticipant_register_contentfilter (p. 120). Can be NULL.

Parameters

<i>writer_filter_data</i>	<< <i>in</i> >> (p. 1013) A pointer to the state created using DDS_ContentFilter::writer_attach (p. 1577) .
<i>prop</i>	<< <i>out</i> >> (p. 1013) A pointer to DDS_ExpressionProperty (p. 1795) that allows you to indicate to RTI Connexx if a filter expression can be optimized.
<i>expression</i>	<< <i>in</i> >> (p. 1013) An ASCIIZ string with the filter expression. The memory used by this string is owned by RTI Connexx and must not be freed. If you want to manipulate this string, you must first make a copy of it.
<i>parameters</i>	<< <i>in</i> >> (p. 1013) A string sequence with the expression parameters with which the DDS_ContentFilteredTopic (p. 177) was created. The string sequence is equal (but not identical) to the string sequence passed to DDS_DomainParticipant_create_contentfilteredtopic (p. 118). Note that the sequence passed to the compile function is owned by RTI Connexx and must not be referenced outside the compile function.
<i>type_code</i>	<< <i>in</i> >> (p. 1013) A pointer to the type code for the related DDS_Topic (p. 175) of the DDS_ContentFilteredTopic (p. 177). A type_code is a description of a type in terms of which types it contains (such as long, 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.
<i>type_class_name</i>	<< <i>in</i> >> (p. 1013) Fully qualified class name of the related DDS_Topic (p. 175).
<i>cookie</i>	<< <i>in</i> >> (p. 1013) DDS_Cookie_t (p. 1580) to uniquely identify DDS_DataReader (p. 615) for which DDS_ContentFilter::writer_compile (p. 1573) was called.

Returns

One of the **Standard Return Codes** (p. 1221)

5.7.3.13 DDS_ContentFilterWriterEvaluateFunction

```
typedef struct DDS_CookieSeq *(* DDS_ContentFilterWriterEvaluateFunction) (void *filter_data,
void *writer_filter_data, const void *sample, const struct DDS_FilterSampleInfo *meta_data)
```

Prototype of **DDS_ContentFilter::writer_evaluate** (p. 1576).

A writer-side filtering API to retrieve a list of DataReaders whose content filters pass the sample.

This function is called every time a **DDS_DataWriter** (p. 485) writes a new sample. Its purpose is to evaluate the sample for all the readers for which the **DDS_DataWriter** (p. 485) is performing writer-side filtering and return the list of **DDS_Cookie_t** (p. 1580) 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

<i>filter_data</i>	<< <i>in</i> >> (p. 1013) The opaque pointer that the content filter was registered with in DDS_DomainParticipant_register_contentfilter (p. 120). Can be NULL.

Parameters

<i>writer_filter_data</i>	<< <i>in</i> >> (p. 1013) A pointer to the state created using DDS_ContentFilter::writer_attach (p. 1577) .
<i>sample</i>	<< <i>in</i> >> (p. 1013) Pointer to a deserialized sample to be filtered.
<i>meta_data</i>	<< <i>in</i> >> (p. 1013) Pointer to meta data associated with the sample.

Returns

The function returns **DDS_CookieSeq** (p. 1581) which identifies the set of DataReaders whose filters pass the sample.

5.7.3.14 DDS_ContentFilterWriterReturnLoanFunction

```
typedef void(* DDS_ContentFilterWriterReturnLoanFunction) (void *filter_data, void *writer_↵
filter_data, struct DDS_CookieSeq *cookies)
```

Prototype of **DDS_ContentFilter::writer_return_loan** (p. 1578).

A writer-side filtering API to return the loan on the list of DataReaders returned by **DDS_ContentFilter::writer_evaluate** (p. 1576).

This function is called to return the loan on **DDS_CookieSeq** (p. 1581) returned by **DDS_ContentFilter::writer_↔
return_loan** (p. 1578). It is possible for multiple threads to be calling into this function at the same time.

Parameters

<i>filter_data</i>	<< <i>in</i> >> (p. 1013) The opaque pointer that the content filter was registered with in DDS_DomainParticipant_register_contentfilter (p. 120). Can be NULL.

Parameters

<i>writer_filter_data</i>	<< <i>in</i> >> (p. 1013) A pointer to the state created using DDS_ContentFilter::writer_attach (p. 1577).
<i>cookies</i>	<< <i>in</i> >> (p. 1013) DDS_CookieSeq (p. 1581) for which the DDS_ContentFilter::writer_return_loan (p. 1578) was invoked.

5.7.3.15 DDS_MultiTopic

```
typedef struct DDS_MultiTopicImpl DDS_MultiTopic
```

[Not supported (optional)] <<*interface*>> (p. 1013) A specialization of **DDS_TopicDescription** (p. 175) that allows subscriptions that combine/filter/rearrange data coming from several topics.

DDS_MultiTopic (p. 185) 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).
- **DDS_DataReader** (p. 615) entities associated with a **DDS_MultiTopic** (p. 185) are alerted of data modifications by the usual **DDS_Listener** (p. 1816) or **DDS_WaitSet** (p. 1371) / **DDS_Condition** (p. 1370) mechanisms whenever modifications occur to the data associated with any of the topics relevant to the **DDS_MultiTopic** (p. 185).

Note that the source for data may not be restricted to a single topic.

DDS_DataReader (p. 615) entities associated with a **DDS_MultiTopic** (p. 185) may access instances that are "constructed" at the **DDS_DataReader** (p. 615) side from the instances written by multiple **DDS_DataWriter** (p. 485) entities. The **DDS_MultiTopic** (p. 185) access instance will begin to exist as soon as all the constituting **DDS_Topic** (p. 175) 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 **DDS_MultiTopic** (p. 185) instance is **DDS_NEW_VIEW_STATE** (p. 713) if at least one of the constituting instances has `view_state = DDS_NEW_VIEW_STATE` (p. 713). Otherwise, it will be **DDS_NOT_NEW_VIEW_STATE** (p. 713).
- The `instance_state` of the **DDS_MultiTopic** (p. 185) instance is **DDS_ALIVE_INSTANCE_STATE** (p. 715) if the `instance_state` of all the constituting **DDS_Topic** (p. 175) instances is **DDS_ALIVE_INSTANCE_STATE** (p. 715). It is **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 715) if at least one of the constituting **DDS_Topic** (p. 175) instances is **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 715). Otherwise, it is **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 715).

Queries and Filters Syntax (p. 738) describes the syntax of `subscription_expression` and `expression`←
_parameters.

5.7.4 Enumeration Type Documentation

5.7.4.1 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 Connex 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.

5.7.5 Function Documentation

5.7.5.1 DDS_InconsistentTopicStatus_initialize()

```
DDS_ReturnCode_t DDS_InconsistentTopicStatus_initialize (
    struct DDS_InconsistentTopicStatus * self )
```

Initializer for new status instances.

New **DDS_InconsistentTopicStatus** (p. 1808) instances stored in heap memory should be initialized with this function before they are passed to any function. This step ensures that those contained fields that use dynamic memory are properly initialized. This function does not allocate memory.

DDS_InconsistentTopicStatus_finalize (p. 188) should be called to free the contained fields that use dynamic memory:

```
DDS_InconsistentTopicStatus *myStatus = malloc(sizeof(struct DDS_InconsistentTopicStatus));

DDS_InconsistentTopicStatus_initialize(myStatus);

DDS_Topic_get_inconsistent_topic_status(myTopic, myStatus);

DDS_InconsistentTopicStatus_finalize(myStatus);

free(myStatus);
```

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_InconsistentTopicStatus_INITIALIZER (p. 173)

DDS_Topic_get_inconsistent_topic_status (p. 196)

DDS_InconsistentTopicStatus_finalize (p. 188)

5.7.5.2 DDS_InconsistentTopicStatus_copy()

```
DDS_ReturnCode_t DDS_InconsistentTopicStatus_copy (
    struct DDS_InconsistentTopicStatus * self,
    const struct DDS_InconsistentTopicStatus * source )
```

Copy the contents of the given status into this status.

Status instances can use dynamic memory because of the sequences contained in some status. A shallow copy by assignment is therefore unsafe. This function performs a deep copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). Status to be copied from.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_InconsistentTopicStatus_INITIALIZER (p. 173)

DDS_InconsistentTopicStatus_initialize (p. 186)

DDS_InconsistentTopicStatus_finalize (p. 188)

5.7.5.3 DDS_InconsistentTopicStatus_finalize()

```
DDS_ReturnCode_t DDS_InconsistentTopicStatus_finalize (
    struct DDS_InconsistentTopicStatus * self )
```

Free any dynamic memory allocated by status instances.

Some status may use dynamic memory (regardless of whether the status itself is in dynamic memory). This function frees that memory but otherwise leaves this status unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

Note that if this status instance is stored in heap memory, calling this function will *not* call free() on it; the user is responsible for explicitly freeing any heap-based status instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_InconsistentTopicStatus_INITIALIZER (p. 173)

DDS_InconsistentTopicStatus_initialize (p. 186)

5.7.5.4 DDS_InconsistentTopicStatus_equals()

```
DDS_Boolean DDS_InconsistentTopicStatus_equals (
    const struct DDS_InconsistentTopicStatus * left,
    const struct DDS_InconsistentTopicStatus * right )
```

Compares two **DDS_InconsistentTopicStatus** (p. 1808) for equality.

Parameters

<i>left</i>	<< <i>in</i> >> (p. 1013) This InconsistentTopicStatus. Can be NULL.
<i>right</i>	<< <i>in</i> >> (p. 1013) The other InconsistentTopicStatus to be compared with this status instance. Can be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two InconsistentTopicStatus have equal values, or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.7.5.5 DDS_TopicQos_equals()

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

Compares two **DDS_TopicQos** (p. 2042) for equality.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) This TopicQos.
<i>other</i>	<< <i>in</i> >> (p. 1013) The other TopicQos to be compared with this TopicQos

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two Qos objects are equal or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.7.5.6 DDS_TopicQos_print()

```
DDS_ReturnCode_t DDS_TopicQos_print (
    const struct DDS_TopicQos * self )
```

Prints this **DDS_TopicQos** (p. 2042) to stdout.

Only the differences between this **DDS_TopicQos** (p. 2042) and the documented default are printed. If you wish to print everything regardless, see **DDS_TopicQos_to_string_w_params** (p. 190). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1926).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_RETCODE_OK (p. 1222) if no error was encountered.

5.7.5.7 DDS_TopicQos_to_string()

```
DDS_ReturnCode_t DDS_TopicQos_to_string (
    const struct DDS_TopicQos * self,
    char * string,
    DDS_UnsignedLong * string_size )
```

Obtains a string representation of this **DDS_TopicQos** (p. 2042).

Only the differences between this **DDS_TopicQos** (p. 2042) and the documented default are printed to the string. If you wish to print everything regardless, see **DDS_TopicQos_to_string_w_params** (p. 190). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1926).

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. 2042) is written to the buffer.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>string</i>	<< <i>out</i> >> (p. 1013) The char buffer that will be used to store the string representation of this DDS_TopicQos (p. 2042). If NULL, this function will return the required length of this buffer through the <code>string_size</code> parameter.
<i>string_size</i>	<< <i>inout</i> >> (p. 1013) 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. 1222) if no error was encountered.

See also

DDS_TopicQos_to_string_w_params (p. 190)

5.7.5.8 DDS_TopicQos_to_string_w_params()

```
DDS_ReturnCode_t DDS_TopicQos_to_string_w_params (
    const struct DDS_TopicQos * self,
```

```
char * string,
    DDS_UnsignedLong * string_size,
    const struct DDS_TopicQos * base,
    const struct DDS_QosPrintFormat * format )
```

Obtains a string representation of this **DDS_TopicQos** (p. 2042).

Only the differences between this **DDS_TopicQos** (p. 2042) and the **DDS_TopicQos** (p. 2042) supplied as the base are printed to the string.

It is possible to supply the sentinel value **DDS_TOPIC_QOS_PRINT_ALL** (p. 164) as the base to print everything within the QoS.

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. 2042) is written to the buffer.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>string</i>	<< <i>out</i> >> (p. 1013) The char buffer that will be used to store the string representation of this DDS_TopicQos (p. 2042). If NULL, this function will return the required length of this buffer through the <code>string_size</code> parameter.
<i>string_size</i>	<< <i>inout</i> >> (p. 1013) 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.
<i>base</i>	<< <i>in</i> >> (p. 1013) The DDS_TopicQos (p. 2042) to be used as the base QoS profile. Only the differences with respect to this base profile will be included in the output string.
<i>format</i>	<< <i>in</i> >> (p. 1013) The DDS_QosPrintFormat (p. 1926) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 1222) if no error was encountered.

5.7.5.9 DDS_TopicQos_initialize()

```
DDS_ReturnCode_t DDS_TopicQos_initialize (
    struct DDS_TopicQos * self )
```

Initializer for new QoS instances.

New **DDS_TopicQos** (p. 2042) instances on heap should be initialized with this function before they are passed to any function. This step ensures that those contained QoS policies that use dynamic memory are properly initialized. This function does not allocate memory.

Calling this function is not a substitute for calling **DDS_Topic_get_qos** (p. 199) or **DDS_DomainParticipant_get_↵ default_topic_qos** (p. 86); one of those should be called subsequently to setting the QoS of any new or existing entity. **DDS_TopicQos_finalize** (p. 192) should be called to free the contained QoS policies that use dynamic memory:

```

DDS_TopicQos *myQos = malloc(sizeof(struct DDS_TopicQos));
DDS_TopicQos_initialize(myQos);
DDS_DomainParticipantFactory_get_default_topic_qos(myFactory, myQos);
DDS_Topic_set_qos(myTopic, myQos);
DDS_TopicQos_finalize(myQos);
free(myQos);

```

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_TopicQos_INITIALIZER (p. 173)

DDS_DomainParticipant_get_default_topic_qos (p. 86)

DDS_TopicQos_finalize (p. 192)

5.7.5.10 DDS_TopicQos_finalize()

```

DDS_ReturnCode_t DDS_TopicQos_finalize (
    struct DDS_TopicQos * self )

```

Free any dynamic memory allocated by the policies in this **DDS_TopicQos** (p. 2042).

Some QoS policies may use dynamic memory (regardless of whether the QoS itself is in dynamic memory). This function frees that memory but otherwise leaves this QoS unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

This function does not leave this object in an invalid state. It is permissible to clear a QoS and then subsequently allocate new dynamic memory in one or more of its QoS policies.

Note that if this QoS instance is stored in heap memory, calling this function will *not* call free() on it; the user is responsible for explicitly freeing any heap-based QoS instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_TopicQos_INITIALIZER (p. 173)

DDS_TopicQos_initialize (p. 191)

5.7.5.11 DDS_TopicQos_copy()

```
DDS_ReturnCode_t DDS_TopicQos_copy (
    struct DDS_TopicQos * self,
    const struct DDS_TopicQos * source )
```

Copy the contents of the given QoS into this QoS.

DDS_TopicQos (p. 2042) instances can use dynamic memory because of the sequences contained in some QoS policies. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). QoS to be copied from.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_TopicQos_INITIALIZER (p. 173)

DDS_TopicQos_initialize (p. 191)

DDS_TopicQos_finalize (p. 192)

5.7.5.12 DDS_TopicDescription_get_type_name()

```
const char * DDS_TopicDescription_get_type_name (
    DDS_TopicDescription * self )
```

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. 221) function.

Thus, the `type_name` implies an association with a corresponding **DDS_TypeSupport** (p. 214) and this **DDS_TopicDescription** (p. 175).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

the type name. The returned type name is valid until the **DDS_TopicDescription** (p. 175) is deleted.

Postcondition

The result is non-NULL.

See also

DDS_TypeSupport (p. 214), **FooTypeSupport** (p. 2114)

5.7.5.13 DDS_TopicDescription_get_name()

```
const char * DDS_TopicDescription_get_name (
    DDS_TopicDescription * self )
```

Get the name used to create this **DDS_TopicDescription** (p. 175) .

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

the name used to create this **DDS_TopicDescription** (p. 175). The returned topic name is valid until the **DDS_TopicDescription** (p. 175) is deleted.

Postcondition

The result is non-NULL.

5.7.5.14 DDS_TopicDescription_get_participant()

```
DDS_DomainParticipant * DDS_TopicDescription_get_participant (
    DDS_TopicDescription * self )
```

Get the **DDS_DomainParticipant** (p. 77) to which the **DDS_TopicDescription** (p. 175) belongs.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

The **DDS_DomainParticipant** (p. 77) to which the **DDS_TopicDescription** (p. 175) belongs.

Postcondition

The result is non-NULL.

5.7.5.15 DDS_Topic_as_entity()

```
DDS_Entity * DDS_Topic_as_entity (
    DDS_Topic * topic )
```

Access a **DDS_Topic** (p. 175)'s **DDS_Entity** (p. 1360) supertype instance.

Returns

DDS_Topic (p. 175)'s supertype **DDS_Entity** (p. 1360) instance

5.7.5.16 DDS_Topic_as_topicdescription()

```
DDS_TopicDescription * DDS_Topic_as_topicdescription (
    DDS_Topic * topic )
```

Access a **DDS_Topic** (p. 175)'s **DDS_TopicDescription** (p. 175) supertype instance.

Returns

DDS_Topic (p. 175)'s supertype **DDS_TopicDescription** (p. 175) instance

Examples

HelloWorld_subscriber.c.

5.7.5.17 DDS_Topic_narrow()

```
DDS_Topic * DDS_Topic_narrow (
    DDS_TopicDescription * self )
```

Narrow the given **DDS_TopicDescription** (p. 175) pointer to a **DDS_Topic** (p. 175) pointer.

Returns

DDS_Topic (p. 175) if this **DDS_TopicDescription** (p. 175) is a **DDS_Topic** (p. 175). Otherwise, return NULL.

5.7.5.18 DDS_Topic_narrow_from_entity()

```
DDS_Topic * DDS_Topic_narrow_from_entity (
    DDS_Entity * self )
```

Narrow the given **DDS_Entity** (p. 1360) pointer to a **DDS_Topic** (p. 175) pointer.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_Topic (p. 175) if this **DDS_Entity** (p. 1360) is a **DDS_Topic** (p. 175). Otherwise, return NULL.

5.7.5.19 DDS_Topic_get_inconsistent_topic_status()

```
DDS_ReturnCode_t DDS_Topic_get_inconsistent_topic_status (
    DDS_Topic * self,
    struct DDS_InconsistentTopicStatus * status )
```

Allows the application to retrieve the **DDS_INCONSISTENT_TOPIC_STATUS** (p. 1228) status of a **DDS_Topic** (p. 175).

Retrieve the current **DDS_InconsistentTopicStatus** (p. 1808)

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>status</i>	<< <i>inout</i> >> (p. 1013) Status to be retrieved. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_InconsistentTopicStatus (p. 1808)

5.7.5.20 DDS_Topic_set_qos()

```
DDS_ReturnCode_t DDS_Topic_set_qos (
    DDS_Topic * self,
    const struct DDS_TopicQos * qos )
```

Set the topic QoS.

The **DDS_TopicQos::topic_data** (p. 2043) and **DDS_TopicQos::deadline** (p. 2044), **DDS_TopicQos::latency**↔**_budget** (p. 2044), **DDS_TopicQos::transport_priority** (p. 2045) and **DDS_TopicQos::lifespan** (p. 2045) can be changed. The other policies are immutable.

Parameters

<i>qos</i>	<< <i>in</i> >> (p. 1013) Set of policies to be applied to DDS_Topic (p. 175).

Policies must be consistent. Immutable policies cannot be changed after **DDS_Topic** (p. 175) is enabled. The special value **DDS_TOPIC_QOS_DEFAULT** (p. 160) can be used to indicate that the QoS of the **DDS_Topic** (p. 175) should be changed to match the current default **DDS_TopicQos** (p. 2042) set in the **DDS_DomainParticipant** (p. 77). Cannot be NULL.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 1222) if immutable policy is changed, or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222) if policies are inconsistent

See also

DDS_TopicQos (p. 2042) for rules on consistency among QoS

set_qos (abstract) (p. 1361)

Operations Allowed in Listener Callbacks (p. 1818)

5.7.5.21 DDS_Topic_set_qos_with_profile()

```
DDS_ReturnCode_t DDS_Topic_set_qos_with_profile (
    DDS_Topic * self,
    const char * library_name,
    const char * profile_name )
```

<<*extension*>> (p. 1012) Change the QoS of this topic using the input XML QoS profile.

The **DDS_TopicQos::topic_data** (p. 2043) and **DDS_TopicQos::deadline** (p. 2044), **DDS_TopicQos::latency**↔**_budget** (p. 2044), **DDS_TopicQos::transport_priority** (p. 2045) and **DDS_TopicQos::lifespan** (p. 2045) can be changed. The other policies are immutable.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< <i>in</i> >> (p. 1013) Library name containing the XML QoS profile. If library_name is null RTI Connexnt will use the default library (see DDS_DomainParticipant_set_default_library (p. 101)).
<i>profile_name</i>	<< <i>in</i> >> (p. 1013) XML QoS Profile name. If profile_name is null RTI Connexnt will use the default profile (see DDS_DomainParticipant_set_default_profile (p. 102)).

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 1222) if immutable policy is changed, or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222) if policies are inconsistent

See also

DDS_TopicQos (p. 2042) for rules on consistency among QoS

Operations Allowed in Listener Callbacks (p. 1818)

5.7.5.22 DDS_Topic_get_qos()

```
DDS_ReturnCode_t DDS_Topic_get_qos (
    DDS_Topic * self,
    struct DDS_TopicQos * qos )
```

Get the topic QoS.

This function may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>inout</i> >> (p. 1013) QoS to be filled up. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

get_qos (abstract) (p. 1362)

5.7.5.23 DDS_Topic_set_listener()

```
DDS_ReturnCode_t DDS_Topic_set_listener (
    DDS_Topic * self,
    const struct DDS_TopicListener * l,
    DDS_StatusMask mask )
```

Set the topic listener.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>l</i>	<< <i>in</i> >> (p. 1013) Listener to be installed on entity.
<i>mask</i>	<< <i>in</i> >> (p. 1013) Changes of communication status to be invoked on the listener. See DDS_StatusMask (p. 1226). The callback function on the listener cannot be NULL if the corresponding status is turned on in the mask.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

set_listener (abstract) (p. 1362)

5.7.5.24 DDS_Topic_get_listener()

```
struct DDS_TopicListener DDS_Topic_get_listener (
    DDS_Topic * self )
```

Get the topic listener.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

Existing listener attached to the **DDS_Topic** (p. 175).

See also

DDS_Topic_get_listenerX (p. 200)

get_listener (abstract) (p. 1363)

5.7.5.25 DDS_Topic_get_listenerX()

```
DDS_ReturnCode_t DDS_Topic_get_listenerX (
    DDS_Topic * self,
    struct DDS_TopicListener * listener )
```

<<*extension*>> (p. 1012) Get the topic listener.

An alternative form of `get_listener` that fills in an existing listener structure rather than returning one on the stack.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>listener</i>	<< <i>inout</i> >> (p. 1013) Listener structure to be filled up. Cannot be NULL.

See also

DDS_Topic_get_listener (p. 200)

get_listener (abstract) (p. 1363)

5.7.5.26 DDS_ContentFilteredTopic_as_topicdescription()

```
DDS_TopicDescription * DDS_ContentFilteredTopic_as_topicdescription (
    DDS_ContentFilteredTopic * contentFilteredTopic )
```

Access a **DDS_ContentFilteredTopic** (p. 177)'s supertype instance.

Parameters

<i>contentFilteredTopic</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_ContentFilteredTopic (p. 177)'s supertype **DDS_TopicDescription** (p. 175) instance

5.7.5.27 DDS_ContentFilteredTopic_narrow()

```
DDS_ContentFilteredTopic * DDS_ContentFilteredTopic_narrow (
    DDS_TopicDescription * self )
```

Narrow the given **DDS_TopicDescription** (p. 175) pointer to a **DDS_ContentFilteredTopic** (p. 177) pointer.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_ContentFilteredTopic (p. 177) if this **DDS_TopicDescription** (p. 175) is a **DDS_ContentFilteredTopic** (p. 177). Otherwise, return NULL.

5.7.5.28 DDS_ContentFilteredTopic_get_filter_expression()

```
const char * DDS_ContentFilteredTopic_get_filter_expression (
    DDS_ContentFilteredTopic * self )
```

Get the `filter_expression`.

Return the `filter_expression` associated with the **DDS_ContentFilteredTopic** (p.177). `filter_expression` is either specified on the last successful call to **DDS_ContentFilteredTopic_set_expression** (p.203) or, if that function is never called, the expression specified when the **DDS_ContentFilteredTopic** (p.177) was created.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

the `filter_expression`.

5.7.5.29 DDS_ContentFilteredTopic_get_expression_parameters()

```
DDS_ReturnCode_t DDS_ContentFilteredTopic_get_expression_parameters (
    DDS_ContentFilteredTopic * self,
    struct DDS_StringSeq * parameters )
```

Get the `expression_parameters`.

Return the `expression_parameters` associated with the **DDS_ContentFilteredTopic** (p.177). `expression_parameters` is either specified on the last successful call to **DDS_ContentFilteredTopic_set_expression_parameters** (p.202), **DDS_ContentFilteredTopic_set_expression** (p.203) or, if that function is never called, the parameters specified when the **DDS_ContentFilteredTopic** (p.177) was created.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>parameters</i>	<< <i>inout</i> >> (p. 1013) the filter expression parameters. Cannot be NULL. The memory for the strings in this sequence is managed according to the conventions described in String Conventions (p.1527). In particular, be careful to avoid a situation in which RTI Connexx allocates a string on your behalf and you then reuse that string in such a way that RTI Connexx believes it to have more memory allocated to it than it actually does.

Returns

One of the **Standard Return Codes** (p.1221)

See also

DDS_DomainParticipant_create_contentfilteredtopic (p.118)

DDS_ContentFilteredTopic_set_expression_parameters (p.202)

5.7.5.30 DDS_ContentFilteredTopic_set_expression_parameters()

```
DDS_ReturnCode_t DDS_ContentFilteredTopic_set_expression_parameters (
    DDS_ContentFilteredTopic * self,
    const struct DDS_StringSeq * parameters )
```

Set the `expression_parameters`.

Change the `expression_parameters` associated with the **DDS_ContentFilteredTopic** (p. 177).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>parameters</i>	<< <i>in</i> >> (p. 1013) the filter expression parameters Cannot be NULL. Length of sequence cannot be greater than 100.

Returns

One of the **Standard Return Codes** (p. 1221)

5.7.5.31 DDS_ContentFilteredTopic_set_expression()

```
DDS_ReturnCode_t DDS_ContentFilteredTopic_set_expression (
    DDS_ContentFilteredTopic * self,
    const char * expression,
    const struct DDS_StringSeq * parameters )
```

Set the `filter_expression` and `expression_parameters`.

Changes the `filter_expression` and `expression_parameters` associated with the **DDS_ContentFilteredTopic** (p. 177).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>expression</i>	<< <i>in</i> >> (p. 1013) the filter expression. Cannot be NULL.
<i>parameters</i>	<< <i>in</i> >> (p. 1013) the filter expression parameters Cannot be NULL. Length of sequence cannot be greater than 100.

Returns

One of the **Standard Return Codes** (p. 1221)

5.7.5.32 DDS_ContentFilteredTopic_append_to_expression_parameter()

```
DDS_ReturnCode_t DDS_ContentFilteredTopic_append_to_expression_parameter (
    DDS_ContentFilteredTopic * self,
    const DDS_Long index,
    const char * val )
```

<<**extension**>> (p. 1012) 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 function 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

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>index</i>	<< in >> (p. 1013) The index of the parameter string to be modified. The first index is index 0. When using the DDS_STRINGMATCHFILTER_NAME (p. 164) filter, <i>index must</i> be 0.
<i>val</i>	<< in >> (p. 1013) The string term to be appended to the parameter string.

Returns

One of the **Standard Return Codes** (p. 1221)

5.7.5.33 DDS_ContentFilteredTopic_remove_from_expression_parameter()

```
DDS_ReturnCode_t DDS_ContentFilteredTopic_remove_from_expression_parameter (
    DDS_ContentFilteredTopic * self,
    const DDS_Long index,
    const char * val )
```

<<**extension**>> (p. 1012) 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 function 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 parameter value is:

'IBM,MSFT'

Then `remove_from_expression_parameter(0, "IBM")` would generate the expression:

'MSFT'

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>index</i>	<< <i>in</i> >> (p. 1013) The index of the parameter string to be modified. The first index is index 0. When using the DDS_STRINGMATCHFILTER_NAME (p. 164) filter, <i>index must</i> be 0.
<i>val</i>	<< <i>in</i> >> (p. 1013) The string term to be removed from the parameter string.

Returns

One of the **Standard Return Codes** (p. 1221)

5.7.5.34 DDS_ContentFilteredTopic_get_related_topic()

```
DDS_Topic * DDS_ContentFilteredTopic_get_related_topic (
    DDS_ContentFilteredTopic * self )
```

Get the `related_topic`.

Return the **DDS_Topic** (p. 175) specified when the **DDS_ContentFilteredTopic** (p. 177) was created.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

The **DDS_Topic** (p. 175) associated with the **DDS_ContentFilteredTopic** (p. 177).

5.7.5.35 DDS_MultiTopic_as_topicdescription()

```
DDS_TopicDescription * DDS_MultiTopic_as_topicdescription (
    DDS_MultiTopic * multiTopic )
```

Access a **DDS_MultiTopic** (p. 185)'s supertype instance.

Parameters

<i>multiTopic</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_MultiTopic (p. 185)'s supertype **DDS_TopicDescription** (p. 175) instance

5.7.5.36 **DDS_MultiTopic_narrow()**

```
DDS_MultiTopic * DDS_MultiTopic_narrow (
    DDS_TopicDescription * self )
```

Narrow the given **DDS_TopicDescription** (p. 175) pointer to a **DDS_MultiTopic** (p. 185) pointer.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_MultiTopic (p. 185) if this **DDS_TopicDescription** (p. 175) is a **DDS_MultiTopic** (p. 185). Otherwise, return NULL.

5.7.5.37 **DDS_MultiTopic_get_subscription_expression()**

```
const char * DDS_MultiTopic_get_subscription_expression (
    DDS_MultiTopic * self )
```

Get the expression for this **DDS_MultiTopic** (p. 185).

The expressions syntax is described in the DDS specification. It is specified when the **DDS_MultiTopic** (p. 185) is created.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

subscription_expression of the **DDS_MultiTopic** (p. 185).

5.7.5.38 DDS_MultiTopic_get_expression_parameters()

```
DDS_ReturnCode_t DDS_MultiTopic_get_expression_parameters (
    DDS_MultiTopic * self,
    struct DDS_StringSeq * parameters )
```

Get the expression parameters.

The expressions syntax is described in the DDS specification.

The `parameters` is either specified on the last successful call to **DDS_MultiTopic_set_expression_parameters** (p.208), or if **DDS_MultiTopic_set_expression_parameters** (p.208) was never called, the `parameters` specified when the **DDS_MultiTopic** (p.185) was created.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>parameters</i>	<< <i>inout</i> >> (p. 1013) Fill in this sequence with the expression parameters. Cannot be NULL. The memory for the strings in this sequence is managed according to the conventions described in String Conventions (p.1527). In particular, be careful to avoid a situation in which RTI Connexx allocates a string on your behalf and you then reuse that string in such a way that RTI Connexx believes it to have more memory allocated to it than it actually does.

Returns

One of the **Standard Return Codes** (p.1221)

5.7.5.39 DDS_MultiTopic_set_expression_parameters()

```
DDS_ReturnCode_t DDS_MultiTopic_set_expression_parameters (
    DDS_MultiTopic * self,
    const struct DDS_StringSeq * parameters )
```

Set the `expression_parameters`.

Changes the `expression_parameters` associated with the **DDS_MultiTopic** (p.185).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>parameters</i>	<< <i>in</i> >> (p. 1013) the filter expression parameters

Returns

One of the **Standard Return Codes** (p.1221).

5.7.6 Variable Documentation

5.7.6.1 DDS_PRINT_FORMAT_PROPERTY_DEFAULT

```
const struct DDS_PrintFormatProperty DDS_PRINT_FORMAT_PROPERTY_DEFAULT
```

Sentinel constant indicating default values for **DDS_PrintFormatProperty** (p. 1894).

Pass this object instead of your own **DDS_PrintFormatProperty** (p. 1894) object to use the default property values:

```
retCode = FooTypeSupport_data_to_string(
    sample,
    &DDS_PRINT_FORMAT_PROPERTY_DEFAULT);
```

See also

DDS_PrintFormatProperty (p. 1894)

5.7.6.2 DDS_FILTERSAMPLEINFO_DEFAULT

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

Initializer for new **DDS_FilterSampleInfo** (p. 1796).

All new **DDS_FilterSampleInfo** (p. 1796) instances allocated on the stack should be initialized to this value. No memory is allocated.

See also

DDS_FilterSampleInfo (p. 1796)

5.8 FlatData Topic-Types

<<**extension**>> (p. 1012) FlatData Language Binding for IDL topic-types

<<**extension**>> (p. 1012) FlatData Language Binding for IDL topic-types

Warning

The FlatData language binding is available in the Traditional C++ API and in the Modern C++ API. It is not available in this API.

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.

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-zero-copy-examples>.

5.9 Zero Copy Transfer Over Shared Memory

<<*extension*>> (p. 1012) Zero Copy transfer over shared memory

<<*extension*>> (p. 1012) 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.

5.10 User Data Type Support

Defines generic classes and macros to support user data types.

Data Structures

- 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 **FooTypeSupport**
<<*interface*>> (p. 1013) <<*generic*>> (p. 1013) *User data type specific interface.*
- struct **DDS_InstanceHandleSeq**
*Instantiates **FooSeq** (p. 2113) < **DDS_InstanceHandle_t** (p. 214) > .*

Macros

- #define **DDS_TYPESUPPORT_C**(TTypeSupport, TData)
Declares the interface required to support a user data type.
- #define **DDS_DATAWRITER_C**(TDataWriter, TData)
Declares the interface required to support a user data type specific data writer.
- #define **DDS_DATAREADER_C**(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.
- typedef struct DDS_TypeSupportImpl **DDS_TypeSupport**
<<interface>> (p. 1013) An abstract marker interface that has to be specialized for each concrete user data type that will be used by the application.

Functions

- **Foo * FooTypeSupport_create_data** (void)
<<extension>> (p. 1012) Create a data type and initialize it.
- **Foo * FooTypeSupport_create_data_ex** (DDS_Boolean allocatePointers)
<<extension>> (p. 1012) Create a data type and initialize it.
- **Foo * FooTypeSupport_create_data_w_params** (const struct **DDS_TypeAllocationParams_t** *alloc_params)
<<extension>> (p. 1012) Create a data type and initialize it.
- **DDS_ReturnCode_t FooTypeSupport_copy_data** (Foo *dst_data, const Foo *src_data)
<<extension>> (p. 1012) Copy data type.
- **DDS_ReturnCode_t FooTypeSupport_delete_data** (Foo *a_data)
<<extension>> (p. 1012) Destroy a user data type instance.
- **DDS_ReturnCode_t FooTypeSupport_delete_data_ex** (Foo *a_data, DDS_Boolean deletePointers)
<<extension>> (p. 1012) Destroy a user data type instance.
- **DDS_ReturnCode_t FooTypeSupport_delete_data_w_params** (Foo *a_data, const struct **DDS_TypeDeallocationParams_t** *dealloc_params)
<<extension>> (p. 1012) Destroy a user data type instance.
- **DDS_ReturnCode_t FooTypeSupport_initialize_data** (Foo *a_data)
<<extension>> (p. 1012) Initialize data type.
- **DDS_ReturnCode_t FooTypeSupport_initialize_data_ex** (Foo *a_data, DDS_Boolean allocatePointers)
<<extension>> (p. 1012) Initialize data type.
- **DDS_ReturnCode_t FooTypeSupport_finalize_data** (Foo *a_data)
<<extension>> (p. 1012) Finalize data type.
- **DDS_ReturnCode_t FooTypeSupport_finalize_data_ex** (Foo *a_data, DDS_Boolean deletePointers)
<<extension>> (p. 1012) Finalize data type.
- const char * **FooTypeSupport_get_type_name** (void)
Get the default name for this type.
- **DDS_ReturnCode_t FooTypeSupport_register_type** (DDS_DomainParticipant *participant, const char *type_name)
Allows an application to communicate to RTI Connexx the existence of a data type.
- **DDS_ReturnCode_t FooTypeSupport_unregister_type** (DDS_DomainParticipant *participant, const char *type_name)
<<extension>> (p. 1012) Allows an application to unregister a data type from RTI Connexx. After calling unregister_type, no further communication using that type is possible.
- void **FooTypeSupport_print_data** (Foo *a_data)
<<extension>> (p. 1012) Print value of data type to standard out.
- **DDS_ReturnCode_t FooTypeSupport_serialize_data_to_cdr_buffer** (char *buffer, unsigned int *length, const Foo *a_data)
<<extension>> (p. 1012) Serializes the input sample into a CDR buffer of octets.

- **DDS_ReturnCode_t FooTypeSupport_serialize_data_to_cdr_buffer_ex** (char *buffer, unsigned int *length, const **Foo** *a_data, **DDS_DataRepresentationId_t** representation)
<<extension>> (p. 1012) Serializes the input sample into a buffer of octets.
- **DDS_ReturnCode_t FooTypeSupport_deserialize_data_from_cdr_buffer** (**Foo** *sample, const char *buffer, unsigned int length)
<<extension>> (p. 1012) Deserializes a sample from a buffer of octets.
- **DDS_ReturnCode_t FooTypeSupport_data_to_string** (**Foo** *sample, char *str, **DDS_UnsignedLong** *str_size, const **DDS_PrintFormatProperty** *property)
<<extension>> (p. 1012) Transforms a data sample into a human-readable string representation.
- **DDS_TypeCode * FooTypeSupport_get_typecode** (void)
<<extension>> (p. 1012) Retrieves the TypeCode for the Type.
- **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. 228).*

Variables

- const **DDS_InstanceHandle_t DDS_HANDLE_NIL**
The NIL instance handle.

5.10.1 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. 2109) are shown below.

an application data type named **Foo** (p. 2109) "

The macros defined here declare the strongly typed APIs needed to support an arbitrary user defined data of type **Foo** (p. 2109).

See also

the `Code Generator User's Manual`

5.10.2 Macro Definition Documentation

5.10.2.1 DDS_TYPESUPPORT_C

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

Declares the interface required to support a user data type.

Defines:

FooTypeSupport (p. 2114) TypeSupport of type **Foo** (p. 2109), i.e. **FooTypeSupport** (p. 2114)

Examples

HelloWorldSupport.c.

5.10.2.2 DDS_DATAWRITER_C

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

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

Uses:

FooTypeSupport (p. 2114) user data type, **Foo** (p. 2109)

Defines:

FooDataWriter (p. 2113) **DDS_DataWriter** (p. 485) of type **Foo** (p. 2109), i.e. **FooDataWriter** (p. 2113)

5.10.2.3 DDS_DATAREADER_C

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

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

Uses:

FooTypeSupport (p. 2114) user data type, **Foo** (p. 2109)
FooSeq (p. 2113) sequence of user data type, `sequence< :Foo>`

Defines:

FooDataReader (p. 2112) **DDS_DataReader** (p. 615) of type **Foo** (p. 2109), i.e. **FooDataReader** (p. 2112)

See also

FooSeq (p. 2113)

Examples

HelloWorldSupport.c.

5.10.3 Typedef Documentation

5.10.3.1 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 **DDS_Topic** (p. 175) of a certain type.

See also

FooDataWriter_register_instance (p. 491)
DDS_SampleInfo::instance_handle (p. 1990)

5.10.3.2 DDS_TypeSupport

```
typedef struct DDS_TypeSupportImpl DDS_TypeSupport
```

<<**interface**>> (p. 1013) 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. 2114), from a description of the type in IDL.

A **DDS_TypeSupport** (p. 214) must be registered using the **FooTypeSupport_register_type** (p. 221) operation on this type-specific class before it can be used to create **DDS_Topic** (p. 175) objects.

See also

FooTypeSupport (p. 2114)
the Code Generator User's Manual

5.10.4 Function Documentation

5.10.4.1 FooTypeSupport_create_data()

```
Foo * FooTypeSupport_create_data (
    void )
```

<<**extension**>> (p. 1012) 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.

Returns

Newly created data type, or NULL on failure.

See also

FooTypeSupport_delete_data (p. 217)

5.10.4.2 FooTypeSupport_create_data_ex()

```
Foo * FooTypeSupport_create_data_ex (
    DDS_Boolean allocatePointers )
```

<<**extension**>> (p. 1012) 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. 1201), all the references (pointers) in the type are recursively allocated.

Parameters

<i>allocatePointers</i>	<< <i>in</i> >> (p. 1013) Whether or not to recursively allocate pointers.

Returns

Newly created data type, or NULL on failure.

See also

FooTypeSupport_delete_data_ex (p. 217)

5.10.4.3 FooTypeSupport_create_data_w_params()

```
Foo * FooTypeSupport_create_data_w_params (
    const struct DDS_TypeAllocationParams_t * alloc_params )
```

<<*extension*>> (p. 1012) 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

<i>alloc_params</i>	<< <i>in</i> >> (p. 1013) 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. 217)

5.10.4.4 FooTypeSupport_copy_data()

```
DDS_ReturnCode_t FooTypeSupport_copy_data (
    Foo * dst_data,
    const Foo * src_data )
```

<<*extension*>> (p. 1012) Copy data type.

The *generated* implementation of the operation knows how to copy value of a data type.

Parameters

<i>dst_data</i>	<< <i>inout</i> >> (p. 1013) Data type to copy value to. Cannot be NULL.
<i>src_data</i>	<< <i>in</i> >> (p. 1013) Data type to copy value from. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.10.4.5 FooTypeSupport_delete_data()

```
DDS_ReturnCode_t FooTypeSupport_delete_data (
    Foo * a_data )
```

<<**extension**>> (p. 1012) Destroy a user data type instance.

The *generated* implementation of the operation knows how to destroy a data type and return all resources.

Parameters

<i>a_data</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

FooTypeSupport_create_data (p. 215)

5.10.4.6 FooTypeSupport_delete_data_ex()

```
DDS_ReturnCode_t FooTypeSupport_delete_data_ex (
    Foo * a_data,
    DDS_Boolean deletePointers )
```

<<**extension**>> (p. 1012) 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. 1201), all the references (pointers) are destroyed as well.

Parameters

<i>a_data</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>deletePointers</i>	<< <i>in</i> >> (p. 1013) Whether or not to destroy pointers.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

FooTypeSupport_create_data_ex (p. 215)

5.10.4.7 FooTypeSupport_delete_data_w_params()

```
DDS_ReturnCode_t FooTypeSupport_delete_data_w_params (
    Foo * a_data,
    const struct DDS_TypeDeallocationParams_t * dealloc_params )
```

<<*extension*>> (p. 1012) Destroy a user data type instance.

The *generated* implementation of the operation knows how to destroy a data type and return all resources.

By default, all non-NULL pointers and optional members are deleted.

Parameters

<i>a_data</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>dealloc_params</i>	<< <i>in</i> >> (p. 1013) Whether or not to destroy pointers and/or optional members.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

FooTypeSupport_create_data_ex (p. 215)

5.10.4.8 FooTypeSupport_initialize_data()

```
DDS_ReturnCode_t FooTypeSupport_initialize_data (
    Foo * a_data )
```

<<**extension**>> (p. 1012) Initialize data type.

The *generated* implementation of the operation knows how to initialize a data type. This function is typically called to initialize a data type that is allocated on the stack. Calling this function more than once will cause a memory leak.

Parameters

<i>a_data</i>	<< inout >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

FooTypeSupport_finalize_data (p. 220)

5.10.4.9 FooTypeSupport_initialize_data_ex()

```
DDS_ReturnCode_t FooTypeSupport_initialize_data_ex (
    Foo * a_data,
    DDS_Boolean allocatePointers )
```

<<**extension**>> (p. 1012) Initialize data type.

The *generated* implementation of the operation knows how to initialize a data type. This function is typically called to initialize a data type that is allocated on the stack. Calling this function more than once will cause a memory leak.

When *allocatePointers* is **DDS_BOOLEAN_TRUE** (p. 1201), all the references (pointers) in the type are recursively allocated.

Parameters

<i>a_data</i>	<< inout >> (p. 1013) Cannot be NULL.
<i>allocatePointers</i>	<< in >> (p. 1013) Whether or not to recursively allocate pointers.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

FooTypeSupport_finalize_data_ex (p. 220)

5.10.4.10 FooTypeSupport_finalize_data()

```
DDS_ReturnCode_t FooTypeSupport_finalize_data (
    Foo * a_data )
```

<<**extension**>> (p. 1012) Finalize data type.

The *generated* implementation of the operation knows how to finalize a data type. This function is typically called to finalize a data type that has previously been initialized.

Parameters

<i>a_data</i>	<< in >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

FooTypeSupport_initialize_data (p. 218)

5.10.4.11 FooTypeSupport_finalize_data_ex()

```
DDS_ReturnCode_t FooTypeSupport_finalize_data_ex (
    Foo * a_data,
    DDS_Boolean deletePointers )
```

<<**extension**>> (p. 1012) Finalize data type.

The *generated* implementation of the operation knows how to finalize a data type. This function is typically called to finalize a data type that has previously been initialized.

When `deletePointers` is **DDS_BOOLEAN_TRUE** (p. 1201), the memory required by the references (pointers) associated to the type is freed.

Parameters

<i>a_data</i>	<< in >> (p. 1013) Cannot be NULL.
<i>deletePointers</i>	<< in >> (p. 1013) Whether or not to free memory allocated by the pointers.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

FooTypeSupport_initialize_data_ex (p. 219)

5.10.4.12 FooTypeSupport_get_type_name()

```
const char * FooTypeSupport_get_type_name (
    void )
```

Get the default name for this type.

Can be used for calling **FooTypeSupport_register_type** (p. 221) or creating **DDS_Topic** (p. 175)

Returns

default name for this type

See also

FooTypeSupport_register_type (p. 221)

DDS_DomainParticipant_create_topic (p. 115)

5.10.4.13 FooTypeSupport_register_type()

```
DDS_ReturnCode_t FooTypeSupport_register_type (
    DDS_DomainParticipant * participant,
    const char * type_name )
```

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 middle-ware 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 **DDS_TypeSupport** (p. 214) can be registered multiple times with a **DDS_DomainParticipant** (p. 77) using the same or different values for the `type_name`. If `register_type` is called multiple times on the same **DDS_TypeSupport** (p. 214) with the same **DDS_DomainParticipant** (p. 77) and `type_name`, the second (and subsequent) registrations are ignored but the operation returns **DDS_RETCODE_OK** (p. 1222).

Precondition

Cannot use the same `type_name` to register two different **DDS_TypeSupport** (p. 214) with the same **DDS_DomainParticipant** (p. 77), or else the operation will fail and **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) will be returned.

Parameters

<i>participant</i>	<< <i>in</i> >> (p. 1013) the DDS_DomainParticipant (p. 77) to register the data type Foo (p. 2109) with. Cannot be NULL.
<i>type_name</i>	<< <i>in</i> >> (p. 1013) the type name under with the data type Foo (p. 2109) is registered with the participant; this type name is used when creating a new DDS_Topic (p. 175). (See DDS_DomainParticipant_create_topic (p. 115).) The name may not be NULL or longer than 255 characters.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

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

DDS_DomainParticipant_create_topic (p. 115)

5.10.4.14 FooTypeSupport_unregister_type()

```
DDS_ReturnCode_t FooTypeSupport_unregister_type (
    DDS_DomainParticipant * participant,
    const char * type_name )
```

<<**extension**>> (p. 1012) 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 **DDS_Topic** (p. 175) objects referencing the type have been destroyed. If any **DDS_Topic** (p. 175) is associated with the type, the operation will fail with **DDS_RETCODE_ERROR** (p. 1222).

Postcondition

All information about the type is removed from RTI Connext. No further communication using this type is possible.

Parameters

<i>participant</i>	<< <i>in</i> >> (p. 1013) the DDS_DomainParticipant (p. 77) to unregister the data type Foo (p. 2109) from. Cannot be NULL.
<i>type_name</i>	<< <i>in</i> >> (p. 1013) the type name under with the data type Foo (p. 2109) 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.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_BAD_PARAMETER** (p. 1222) or **DDS_RETCODE_ERROR** (p. 1222)

MT Safety:

SAFE.

See also

FooTypeSupport_register_type (p. 221)

5.10.4.15 FooTypeSupport_print_data()

```
void FooTypeSupport_print_data (
    Foo * a_data )
```

<<**extension**>> (p. 1012) Print value of data type to standard out.

The *generated* implementation of the operation knows how to print value of a data type.

Parameters

<i>a_data</i>	<< <i>in</i> >> (p. 1013) Data type to be printed.

5.10.4.16 FooTypeSupport_serialize_data_to_cdr_buffer()

```
DDS_ReturnCode_t FooTypeSupport_serialize_data_to_cdr_buffer (
    char * buffer,
    unsigned int * length,
    const Foo * a_data )
```

<<**extension**>> (p. 1012) Serializes the input sample into a CDR buffer of octets.

This function serializes a sample into a buffer of octets using Common Data Representation (CDR). Calling this function is equivalent to calling **FooTypeSupport_serialize_data_to_cdr_buffer_ex** (p. 224) with **DDS_AUTO_DATA_↵ REPRESENTATION** (p. 1258) as the representation.

The input buffer must be big enough to store the serialized representation of the sample. Otherwise, the function 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 Connex, see [Data Representation](#) in the Core Libraries Extensible Types Guide.

Parameters

<i>a_data</i>	<< in >> (p. 1013). Input sample. Cannot be NULL.
<i>buffer</i>	<< out >> (p. 1013). Serialization buffer.
<i>length</i>	<< inout >> (p. 1013). When <i>buffer</i> is set to NULL, after the function executes, <i>length</i> will contain a buffer size big enough to hold the serialized data. When <i>buffer</i> is not NULL, <i>length</i> must contain the size of the input buffer when the function is invoked. After the function executes, <i>length</i> will be updated to contain the actual size of the serialized content, which may be smaller than the size obtained when <i>buffer</i> is set to NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.10.4.17 FooTypeSupport_serialize_data_to_cdr_buffer_ex()

```
DDS_ReturnCode_t FooTypeSupport_serialize_data_to_cdr_buffer_ex (
    char * buffer,
    unsigned int * length,
    const Foo * a_data,
    DDS_DataRepresentationId_t representation )
```

<<**extension**>> (p. 1012) Serializes the input sample into a buffer of octets.

This function serializes a sample into a buffer of octets using the input data representation. See **FooTypeSupport_↵ serialize_data_to_cdr_buffer** (p. 223) for details.

Parameters

<i>a_data</i>	<< in >> (p. 1013). Input sample. Cannot be NULL.
<i>buffer</i>	<< out >> (p. 1013). Serialization buffer.
<i>length</i>	<< inout >> (p. 1013). Serialization buffer length.
<i>representation</i>	<< in >> (p. 1013). Representation used to serialize the data.

Returns

One of the **Standard Return Codes** (p. 1221)

5.10.4.18 FooTypeSupport_deserialize_data_from_cdr_buffer()

```
DDS_ReturnCode_t FooTypeSupport_deserialize_data_from_cdr_buffer (
    Foo * sample,
    const char * buffer,
    unsigned int length )
```

<<**extension**>> (p. 1012) Deserializes a sample from a buffer of octets.

This function deserializes a sample from a CDR buffer of octets.

The content of the buffer generated by the function **FooTypeSupport_serialize_data_to_cdr_buffer** (p. 223) can be provided to this function to get the sample back.

Parameters

<i>sample</i>	<< in >> (p. 1013). Output sample. Cannot be NULL.
<i>buffer</i>	<< in >> (p. 1013). Deserialization buffer. Cannot be NULL.
<i>length</i>	<< in >> (p. 1013). Length of the serialized representation of the sample in the buffer.

Returns

One of the **Standard Return Codes** (p. 1221)

5.10.4.19 FooTypeSupport_data_to_string()

```
DDS_ReturnCode_t FooTypeSupport_data_to_string (
    Foo * sample,
    char * str,
    DDS_UnsignedLong * str_size,
    const DDS_PrintFormatProperty * property )
```

<<**extension**>> (p. 1012) Transforms a data sample into a human-readable string representation.

This function 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 function 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. 1222).

This method is only provided for types that were generated with typecodes.

Parameters

<i>sample</i>	<< <i>in</i> >> (p. 1013). The sample to get the string representation for. Cannot be NULL.
<i>str</i>	<< <i>out</i> >> (p. 1013). Output string representing the data sample.
<i>str_size</i>	<< <i>inout</i> >> (p. 1013). When <i>str</i> is set to NULL, after the function executes, <i>str_size</i> will contain a buffer size big enough to hold the string representation of the data. When <i>str</i> is not NULL, <i>str_size</i> must contain the size of the input buffer when the function is invoked. If the size of the input buffer is too small, after the function executes, <i>str_size</i> will be updated to contain the required size of the string content and the function will return DDS_RETCODE_OUT_OF_RESOURCES (p. 1222).
<i>property</i>	<< <i>in</i> >> (p. 1013). Properties describing what the format of the output string should be.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

5.10.4.20 FooTypeSupport_get_typecode()

```
DDS_TypeCode * FooTypeSupport_get_typecode (
    void )
```

<<*extension*>> (p. 1012) Retrieves the TypeCode for the Type.

This function retrieves the **DDS_TypeCode** (p.2071) for the Type. A **DDS_TypeCode** (p.2071) 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.2071) value consists of a type code *kind* (represented by the **DDS_TCKind** (p.243) enumeration) and a list of *members* (that is, fields). These members are recursive: each one has its own **DDS_TypeCode** (p.2071), 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

5.10.4.21 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

<i>self</i>	<< <i>in</i> >> (p. 1013) This handle. Cannot be NULL.
<i>other</i>	<< <i>in</i> >> (p. 1013) The other handle to be compared with this handle. Cannot be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two handles have equal values, or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

See also

DDS_InstanceHandle_is_nil (p. 228)

5.10.4.22 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

<i>self</i>	<< <i>in</i> >> (p. 1013) This handle. Cannot be NULL.
<i>other</i>	<< <i>in</i> >> (p. 1013) 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. 228)

5.10.4.23 DDS_InstanceHandle_copy()

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

Copies this instance handle into another handle.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) The source handle. Cannot be NULL.
<i>other</i>	<< <i>in</i> >> (p. 1013) The destination handle to be copied into. Cannot be NULL.

5.10.4.24 DDS_InstanceHandle_is_nil()

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

Compare this handle to **DDS_HANDLE_NIL** (p. 228).

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the given instance handle is equal to **DDS_HANDLE_NIL** (p. 228) or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

See also

DDS_InstanceHandle_equals (p. 226)

5.10.5 Variable Documentation

5.10.5.1 DDS_HANDLE_NIL

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

The NIL instance handle.

Special **DDS_InstanceHandle_t** (p. 214) value

See also

DDS_InstanceHandle_is_nil (p. 228)

Examples

HelloWorld_publisher.c.

5.11 Type Code Support

<<*extension*>> (p. 1012) A **DDS_TypeCode** (p. 2071) 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. 313) capability or to inspect the type information you receive from remote readers and writers.

Data Structures

- 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 rtdsgen (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. 2071) member ID.*
- #define **DDS_TYPECODE_INDEX_INVALID**
*A sentinel indicating an invalid **DDS_TypeCode** (p. 2071) 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. 2073).

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 }
The different formats to print a `DDS_TypeCode` (p. 2071).
- **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. 2071) kinds.

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

Type to indicate the extensibility of a type.

Functions

- DDS_TCKind DDS_TypeCode_kind** (const **DDS_TypeCode** *self, **DDS_ExceptionCode_t** *ex)
*Gets the **DDS_TCKind** (p. 243) value of a type code.*
- DDS_ExtensibilityKind DDS_TypeCode_extensibility_kind** (const **DDS_TypeCode** *self, **DDS_ExceptionCode_t** *ex)
*Gets the **DDS_ExtensibilityKind** (p. 243) value of a type code.*
- DDS_Boolean DDS_TypeCode_equal** (const **DDS_TypeCode** *self, const **DDS_TypeCode** *tc, **DDS_ExceptionCode_t** *ex)
*Compares two **DDS_TypeCode** (p. 2071) objects for equality.*
- const char * **DDS_TypeCode_name** (const **DDS_TypeCode** *self, **DDS_ExceptionCode_t** *ex)
*Retrieves the simple name identifying this **DDS_TypeCode** (p. 2071) object within its enclosing scope.*
- const **DDS_TypeCode_Annotation** * **DDS_TypeCode_default_annotation** (const **DDS_TypeCode** *self, **DDS_ExceptionCode_t** *ex)
[DEPRECATED]** Given a **DDS_TypeCode** (p. 2071) object, retrieves its **[default]** annotation value as a **DDS_TypeCode_Annotation
- const **DDS_TypeCode_Annotation** * **DDS_TypeCode_member_default_annotation** (const **DDS_TypeCode** *self, **DDS_UnsignedLong** index, **DDS_ExceptionCode_t** *ex)
***[DEPRECATED]** Returns the **[default]** annotation of a type code member identified by the given index.*
- const **DDS_TypeCode_Annotation** * **DDS_TypeCode_min_annotation** (const **DDS_TypeCode** *self, **DDS_ExceptionCode_t** *ex)
[DEPRECATED]** Given a **DDS_TypeCode** (p. 2071) object, retrieves its @min annotation value as a **DDS_TypeCode_Annotation
- const **DDS_TypeCode_Annotation** * **DDS_TypeCode_max_annotation** (const **DDS_TypeCode** *self, **DDS_ExceptionCode_t** *ex)
[DEPRECATED]** Given a **DDS_TypeCode** (p. 2071) object, retrieves its @max annotation value as a **DDS_TypeCode_Annotation
- const **DDS_AnnotationParameterValue** * **DDS_TypeCode_default_value** (const **DDS_TypeCode** *self, **DDS_ExceptionCode_t** *ex)
*Returns the default annotation for this **DDS_TypeCode** (p. 2071).*
- const **DDS_AnnotationParameterValue** * **DDS_TypeCode_min_value** (const **DDS_TypeCode** *self, **DDS_ExceptionCode_t** *ex)
*Returns the min annotation for this **DDS_TypeCode** (p. 2071).*
- const **DDS_AnnotationParameterValue** * **DDS_TypeCode_max_value** (const **DDS_TypeCode** *self, **DDS_ExceptionCode_t** *ex)
*Returns the max annotation for this **DDS_TypeCode** (p. 2071).*

- **const DDS_AnnotationParameterValue * DDS_TypeCode_member_default_value** (const DDS_TypeCode *self, DDS_UnsignedLong index, DDS_ExceptionCode_t *ex)
Returns the member default annotation for this DDS_TypeCode (p. 2071).
- **const DDS_AnnotationParameterValue * DDS_TypeCode_member_min_value** (const DDS_TypeCode *self, DDS_UnsignedLong index, DDS_ExceptionCode_t *ex)
Returns the member min annotation for this DDS_TypeCode (p. 2071).
- **const DDS_AnnotationParameterValue * DDS_TypeCode_member_max_value** (const DDS_TypeCode *self, DDS_UnsignedLong index, DDS_ExceptionCode_t *ex)
Returns the member max annotation for this DDS_TypeCode (p. 2071).
- **DDS_UnsignedLong DDS_TypeCode_member_count** (const DDS_TypeCode *self, DDS_ExceptionCode_t *ex)
Returns the number of members of the type code.
- **const char * DDS_TypeCode_member_name** (const DDS_TypeCode *self, DDS_UnsignedLong index, DDS_ExceptionCode_t *ex)
Returns the name of a type code member identified by the given index.
- **DDS_UnsignedLong DDS_TypeCode_find_member_by_name** (const DDS_TypeCode *self, const char *name, DDS_ExceptionCode_t *ex)
Get the index of the member of the given name.
- **DDS_TypeCode * DDS_TypeCode_member_type** (const DDS_TypeCode *self, DDS_UnsignedLong index, DDS_ExceptionCode_t *ex)
Retrieves the DDS_TypeCode (p. 2071) object describing the type of the member identified by the given index.
- **DDS_UnsignedLong DDS_TypeCode_member_label_count** (const DDS_TypeCode *self, DDS_UnsignedLong index, DDS_ExceptionCode_t *ex)
Returns the number of labels associated to the index-th union member.
- **DDS_Long DDS_TypeCode_member_label** (const DDS_TypeCode *self, DDS_UnsignedLong member_index, DDS_UnsignedLong label_index, DDS_ExceptionCode_t *ex)
Return the label_index-th label associated to the member_index-th member.
- **DDS_Long DDS_TypeCode_member_ordinal** (const DDS_TypeCode *self, DDS_UnsignedLong index, DDS_ExceptionCode_t *ex)
Returns the ordinal that corresponds to the index-th enum value.
- **DDS_Boolean DDS_TypeCode_is_member_key** (const DDS_TypeCode *self, DDS_UnsignedLong index, DDS_ExceptionCode_t *ex)
Function that tells if a member is a key or not.
- **DDS_Boolean DDS_TypeCode_is_member_required** (const DDS_TypeCode *self, DDS_UnsignedLong index, DDS_ExceptionCode_t *ex)
Indicates whether a given member of a type is required to be present in every sample of that type.
- **DDS_Boolean DDS_TypeCode_is_member_pointer** (const DDS_TypeCode *self, DDS_UnsignedLong index, DDS_ExceptionCode_t *ex)
Function that tells if a member is a pointer or not.
- **DDS_Boolean DDS_TypeCode_is_member_bitfield** (const DDS_TypeCode *self, DDS_UnsignedLong index, DDS_ExceptionCode_t *ex)
Function that tells if a member is a bitfield or not.
- **DDS_Short DDS_TypeCode_member_bitfield_bits** (const DDS_TypeCode *self, DDS_UnsignedLong index, DDS_ExceptionCode_t *ex)
Returns the number of bits of a bitfield member.
- **DDS_Visibility DDS_TypeCode_member_visibility** (const DDS_TypeCode *self, DDS_UnsignedLong index, DDS_ExceptionCode_t *ex)
Returns the constant that indicates the visibility of the index-th member.

- **DDS_TypeCode * DDS_TypeCode_discriminator_type** (const **DDS_TypeCode** *self, **DDS_ExceptionCode_t** *ex)
Returns the discriminator type code.
- **DDS_UnsignedLong DDS_TypeCode_length** (const **DDS_TypeCode** *self, **DDS_ExceptionCode_t** *ex)
Returns the number of elements in the type described by this type code.
- **DDS_UnsignedLong DDS_TypeCode_array_dimension_count** (const **DDS_TypeCode** *self, **DDS_ExceptionCode_t** *ex)
This function returns the number of dimensions of an array type code.
- **DDS_UnsignedLong DDS_TypeCode_array_dimension** (const **DDS_TypeCode** *self, **DDS_UnsignedLong** index, **DDS_ExceptionCode_t** *ex)
This function returns the index-th dimension of an array type code.
- **DDS_UnsignedLong DDS_TypeCode_element_count** (const **DDS_TypeCode** *self, **DDS_ExceptionCode_t** *ex)
The number of elements in an array.
- **DDS_TypeCode * DDS_TypeCode_content_type** (const **DDS_TypeCode** *self, **DDS_ExceptionCode_t** *ex)
*Returns the **DDS_TypeCode** (p. 2071) object representing the type for the members of the object described by this **DDS_TypeCode** (p. 2071) object.*
- **DDS_Boolean DDS_TypeCode_is_alias_pointer** (const **DDS_TypeCode** *self, **DDS_ExceptionCode_t** *ex)
Function that tells if an alias is a pointer or not.
- **DDS_Long DDS_TypeCode_default_index** (const **DDS_TypeCode** *self, **DDS_ExceptionCode_t** *ex)
Returns the index of the default member, or -1 if there is no default member.
- **DDS_TypeCode * DDS_TypeCode_concrete_base_type** (const **DDS_TypeCode** *self, **DDS_ExceptionCode_t** *ex)
*Returns the **DDS_TypeCode** (p. 2071) that describes the concrete base type of the value type that this **DDS_TypeCode** (p. 2071) object describes.*
- **DDS_ValueModifier DDS_TypeCode_type_modifier** (const **DDS_TypeCode** *self, **DDS_ExceptionCode_t** *ex)
*Returns a constant indicating the modifier of the value type that this **DDS_TypeCode** (p. 2071) object describes.*
- **DDS_UnsignedLong DDS_TypeCode_find_member_by_id** (const **DDS_TypeCode** *self, **DDS_Long** id, **DDS_ExceptionCode_t** *ex)
Get the index of the member of the given ID.
- **DDS_Long DDS_TypeCode_member_id** (const **DDS_TypeCode** *self, **DDS_UnsignedLong** index, **DDS_ExceptionCode_t** *ex)
Returns the ID of the TypeCode member identified by the given index.
- **DDS_UnsignedLong DDS_TypeCode_get_type_object_serialized_size** (const **DDS_TypeCode** *self, **DDS_ExceptionCode_t** *ex)
*Gets the serialized size of the TypeObject created from this **DDS_TypeCode** (p. 2071).*
- **DDS_UnsignedLong DDS_TypeCode_add_member_to_enum** (**DDS_TypeCode** *self, const char *name, **DDS_Long** ordinal, **DDS_ExceptionCode_t** *ex)
*Add a new enumerated constant to this enum **DDS_TypeCode** (p. 2071).*
- **DDS_UnsignedLong DDS_TypeCode_add_member_to_union** (**DDS_TypeCode** *self, const char *name, **DDS_Long** id, const struct **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. 2071).*
- **DDS_UnsignedLong DDS_TypeCode_add_member** (**DDS_TypeCode** *self, 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. 2071).*

- **DDS_UnsignedLong DDS_TypeCode_add_member_ex** (DDS_TypeCode *self, 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_ExceptionCode_t *ex)
Add a new member to this DDS_TypeCode (p. 2071).
- **void DDS_TypeCode_print_IDL** (const DDS_TypeCode *self, DDS_UnsignedLong indent, DDS_↵
ExceptionCode_t *ex)
Prints a DDS_TypeCode (p. 2071) in IDL notation.
- **void DDS_TypeCode_print** (const DDS_TypeCode *self, const DDS_TypeCodePrintFormatProperty
*format, DDS_ExceptionCode_t *ex)
Prints a DDS_TypeCode (p. 2071).
- **void DDS_TypeCode_to_string** (const DDS_TypeCode *self, char *str, DDS_UnsignedLong *str_size,
DDS_ExceptionCode_t *ex)
*Get a string representation of this DDS_TypeCode (p. 2071) object using the default values for DDS_TypeCodePrint↵
FormatProperty (p. 2073).*
- **void DDS_TypeCode_to_string_w_format** (const DDS_TypeCode *self, char *str, DDS_UnsignedLong
*str_size, const DDS_TypeCodePrintFormatProperty *format, DDS_ExceptionCode_t *ex)
*Get a string representation of this DDS_TypeCode (p. 2071) object using the format described by DDS_TypeCode↵
PrintFormatProperty (p. 2073).*
- **DDS_UnsignedLong DDS_TypeCode_get_cdr_serialized_sample_max_size** (const DDS_TypeCode *self,
DDS_DataRepresentationId_t representation_id, DDS_ExceptionCode_t *ex)
[DEPRECATED] Gets the maximum serialized size of samples of this type
- **DDS_UnsignedLong DDS_TypeCode_cdr_serialized_sample_max_size** (const DDS_TypeCode *self,
DDS_DataRepresentationId_t representation_id, DDS_ExceptionCode_t *ex)
Gets the maximum serialized size of samples of this type.
- **DDS_UnsignedLong DDS_TypeCode_cdr_serialized_sample_min_size** (const DDS_TypeCode *self,
DDS_DataRepresentationId_t representation_id, DDS_ExceptionCode_t *ex)
Gets the minimum serialized size of samples of this type.
- **DDS_UnsignedLong DDS_TypeCode_cdr_serialized_sample_key_max_size** (const DDS_TypeCode *self,
DDS_DataRepresentationId_t representation_id, DDS_ExceptionCode_t *ex)
Gets the maximum serialized size of sample keys of this type.
- **DDS_TypeCodeFactory * DDS_TypeCodeFactory_get_instance** (void)
Gets the singleton instance of this class.
- **DDS_ReturnCode_t DDS_TypeCodeFactory_finalize_instance** (void)
*If there are no DDS_DomainParticipantFactory instances left in the system, it will finalize the DDS_TypeCodeFactory
(p. 2071). If there is at least one DDS_DomainParticipantFactory instance left in the system, this is a noop.*
- **DDS_TypeCode * DDS_TypeCodeFactory_clone_tc** (DDS_TypeCodeFactory *self, const DDS_TypeCode
*tc, DDS_ExceptionCode_t *ex)
Creates and returns a copy of the input DDS_TypeCode (p. 2071).
- **void DDS_TypeCodeFactory_delete_tc** (DDS_TypeCodeFactory *self, DDS_TypeCode *tc, DDS_↵
ExceptionCode_t *ex)
Deletes the input DDS_TypeCode (p. 2071).
- **const DDS_TypeCode * DDS_TypeCodeFactory_get_primitive_tc** (DDS_TypeCodeFactory *self, DDS_↵
_TCKind tc_kind)
*Get the DDS_TypeCode (p. 2071) for a primitive type (integers, floating point values, etc.) identified by the given DDS_↵
_TCKind (p. 243).*
- **DDS_TypeCode * DDS_TypeCodeFactory_create_struct_tc** (DDS_TypeCodeFactory *self, const char
*name, const struct DDS_StructMemberSeq *members, DDS_ExceptionCode_t *ex)
Constructs a DDS_TK_STRUCT (p. 243) DDS_TypeCode (p. 2071).

- **DDS_TypeCode * DDS_TypeCodeFactory_create_struct_tc_ex** (DDS_TypeCodeFactory *self, const char *name, DDS_ExtensibilityKind extensibility_kind, const struct DDS_StructMemberSeq *members, DDS_↵ ExceptionCode_t *ex)
Constructs a DDS_TK_STRUCTURE (p. 243) DDS_TypeCode (p. 2071).
- **DDS_TypeCode * DDS_TypeCodeFactory_create_value_tc** (DDS_TypeCodeFactory *self, const char *name, DDS_ValueModifier type_modifier, const DDS_TypeCode *concrete_base, const struct DDS_Value↵ MemberSeq *members, DDS_ExceptionCode_t *ex)
Constructs a DDS_TK_VALUE (p. 243) DDS_TypeCode (p. 2071).
- **DDS_TypeCode * DDS_TypeCodeFactory_create_value_tc_ex** (DDS_TypeCodeFactory *self, const char *name, DDS_ExtensibilityKind extensibility_kind, DDS_ValueModifier type_modifier, const DDS_TypeCode *concrete_base, const struct DDS_ValueMemberSeq *members, DDS_ExceptionCode_t *ex)
Constructs a DDS_TK_VALUE (p. 243) DDS_TypeCode (p. 2071).
- **DDS_TypeCode * DDS_TypeCodeFactory_create_union_tc** (DDS_TypeCodeFactory *self, const char *name, const DDS_TypeCode *discriminator_type, DDS_Long default_index, const struct DDS_Union↵ MemberSeq *members, DDS_ExceptionCode_t *ex)
Constructs a DDS_TK_UNION (p. 243) DDS_TypeCode (p. 2071).
- **DDS_TypeCode * DDS_TypeCodeFactory_create_union_tc_ex** (DDS_TypeCodeFactory *self, const char *name, DDS_ExtensibilityKind extensibility_kind, const DDS_TypeCode *discriminator_type, DDS_Long default_index, const struct DDS_UnionMemberSeq *members, DDS_ExceptionCode_t *ex)
Constructs a DDS_TK_UNION (p. 243) DDS_TypeCode (p. 2071).
- **DDS_TypeCode * DDS_TypeCodeFactory_create_enum_tc** (DDS_TypeCodeFactory *self, const char *name, const struct DDS_EnumMemberSeq *members, DDS_ExceptionCode_t *ex)
Constructs a DDS_TK_ENUM (p. 243) DDS_TypeCode (p. 2071).
- **DDS_TypeCode * DDS_TypeCodeFactory_create_enum_tc_ex** (DDS_TypeCodeFactory *self, const char *name, DDS_ExtensibilityKind extensibility_kind, const struct DDS_EnumMemberSeq *members, DDS_↵ ExceptionCode_t *ex)
Constructs a DDS_TK_ENUM (p. 243) DDS_TypeCode (p. 2071).
- **DDS_TypeCode * DDS_TypeCodeFactory_create_alias_tc** (DDS_TypeCodeFactory *self, const char *name, const DDS_TypeCode *original_type, DDS_Boolean is_pointer, DDS_ExceptionCode_t *ex)
Constructs a DDS_TK_ALIAS (p. 243) (typedef) DDS_TypeCode (p. 2071).
- **DDS_TypeCode * DDS_TypeCodeFactory_create_string_tc** (DDS_TypeCodeFactory *self, DDS_↵ UnsignedLong bound, DDS_ExceptionCode_t *ex)
Constructs a DDS_TK_STRING (p. 243) DDS_TypeCode (p. 2071).
- **DDS_TypeCode * DDS_TypeCodeFactory_create_wstring_tc** (DDS_TypeCodeFactory *self, DDS_↵ UnsignedLong bound, DDS_ExceptionCode_t *ex)
Constructs a DDS_TK_WSTRING (p. 243) DDS_TypeCode (p. 2071).
- **DDS_TypeCode * DDS_TypeCodeFactory_create_sequence_tc** (DDS_TypeCodeFactory *self, DDS_↵ UnsignedLong bound, const DDS_TypeCode *element_type, DDS_ExceptionCode_t *ex)
Constructs a DDS_TK_SEQUENCE (p. 243) DDS_TypeCode (p. 2071).
- **DDS_TypeCode * DDS_TypeCodeFactory_create_array_tc** (DDS_TypeCodeFactory *self, const struct DDS_UnsignedLongSeq *dimensions, const DDS_TypeCode *element_type, DDS_ExceptionCode_t *ex)
Constructs a DDS_TK_ARRAY (p. 243) DDS_TypeCode (p. 2071).

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.

5.11.1 Detailed Description

<<**extension**>> (p. 1012) A **DDS_TypeCode** (p. 2071) 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. 313) 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. 2071) class, which has an analogue in CORBA.

A **DDS_TypeCode** (p. 2071) value consists of a type code *kind* (represented by the **DDS_TCKind** (p. 243) enumeration) and a list of *members* (that is, fields). These members are recursive: each one has its own **DDS_TypeCode** (p. 2071), 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. 247) function is a more reliable test than comparing the string type names, requiring equivalent definitions of the types.

5.11.2 Accessing a Local ::DDS_TypeCode

When generating types with `rtiddsgen`, type codes are always enabled. For these types, a **DDS_TypeCode** (p. 2071) may be accessed by calling the `Foo_get_typecode` function for a type "Foo", which returns a **DDS_TypeCode** (p. 2071) pointer. via the **FooTypeCode.VALUE** member. This API also includes support for dynamic creation of **DDS_TypeCode** (p. 2071) values, typically for use with the **Dynamic Data** (p. 313) API. You can create a **DDS_TypeCode** (p. 2071) using the **DDS_TypeCodeFactory** (p. 2071) class. You will construct the **DDS_TypeCode** (p. 2071) 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. 291), which provides the **DDS_TypeCode** (p. 2071) instances corresponding to the primitive types (e.g. **DDS_TK_LONG** (p. 243), **DDS_TK_SHORT** (p. 243), and so on).
- **DDS_TypeCodeFactory_create_string_tc** (p. 298) and **DDS_TypeCodeFactory_create_wstring_tc** (p. 298) create a **DDS_TypeCode** (p. 2071) representing a text string with a certain *bound* (i.e. maximum length).
- **DDS_TypeCodeFactory_create_array_tc** (p. 300) and **DDS_TypeCodeFactory_create_sequence_tc** (p. 299) create a **DDS_TypeCode** (p. 2071) for a collection based on the **DDS_TypeCode** (p. 2071) for its elements.
- **DDS_TypeCodeFactory_create_struct_tc** (p. 291) and **DDS_TypeCodeFactory_create_value_tc** (p. 293) create a **DDS_TypeCode** (p. 2071) for a structured type.

5.11.3 Accessing a Remote ::DDS_TypeCode

In addition to being used locally, RTI Connext can transmit **DDS_TypeCode** (p. 2071) 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. 313)). This functionality is useful for a generic system monitoring tool like `rtiddsspy`.

Remote **DDS_TypeCode** (p. 2071) 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. 165). Discovered **DDS_TypeCode** (p. 2071) values are not cached by RTI Connext upon receipt and are therefore not available from the built-in topic data returned by **DDS_DataWriter_get_matched_subscription_data** (p. 540) or **DDS_DataReader_get_matched_publication_data** (p. 679).

The space available locally to deserialize a discovered remote **DDS_TypeCode** (p. 2071) is specified by the **DDS_DomainParticipant** (p. 77)'s **DDS_DomainParticipantResourceLimitsQosPolicy::type_code_max_serialized_length** (p. 1749) 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. 2071)
Dynamic Data (p. 313)
the `Code Generator User's Manual`
DDS_SubscriptionBuiltinTopicData (p. 2012)
DDS_PublicationBuiltinTopicData (p. 1905)

5.11.4 Macro Definition Documentation

5.11.4.1 DDS_TYPECODE_MEMBER_ID_INVALID

```
#define DDS_TYPECODE_MEMBER_ID_INVALID
```

A sentinel indicating an invalid **DDS_TypeCode** (p. 2071) member ID.

5.11.4.2 DDS_TYPECODE_INDEX_INVALID

```
#define DDS_TYPECODE_INDEX_INVALID
```

A sentinel indicating an invalid **DDS_TypeCode** (p. 2071) member index.

5.11.4.3 DDS_TYPECODE_NOT_BITFIELD

```
#define DDS_TYPECODE_NOT_BITFIELD
```

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

5.11.4.4 DDS_VM_NONE

```
#define DDS_VM_NONE
```

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

See also

DDS_ValueModifier (p. 242)

Examples

HelloWorld.c.

5.11.4.5 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. 242)

5.11.4.6 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. 242)

5.11.4.7 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. 242)

5.11.4.8 DDS_PRIVATE_MEMBER

```
#define DDS_PRIVATE_MEMBER
```

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

See also

DDS_Visibility (p. 242)

DDS_PUBLIC_MEMBER (p. 240)

5.11.4.9 DDS_PUBLIC_MEMBER

```
#define DDS_PUBLIC_MEMBER
```

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

See also

DDS_Visibility (p. 242)

DDS_PRIVATE_MEMBER (p. 239)

Examples

HelloWorld.c.

5.11.4.10 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. 313) facility, a **DDS_DynamicData** (p. 1768) sample of the type will only contain a value for a **DDS_TYPECODE_NONKEY_MEMBER** (p. 240) field if one has been explicitly set (see, for example, **DDS_DynamicData_set_long** (p. 394)). The middleware will *not* assume any default value.

See also

DDS_TYPECODE_KEY_MEMBER (p. 240)

DDS_TYPECODE_NONKEY_REQUIRED_MEMBER (p. 241)

DDS_TYPECODE_KEY_MEMBER (p. 240)

DDS_TypeCode_add_member (p. 280)

DDS_TypeCode_add_member_ex (p. 282)

DDS_TypeCode_is_member_key (p. 261)

DDS_TypeCode_is_member_required (p. 262)

DDS_StructMember::is_key (p. 2008)

DDS_ValueMember::is_key (p. 2089)

5.11.4.11 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. 313) facility, all **DDS_DynamicData** (p. 1768) samples of the type will contain a value for all **DDS_TYPECODE_KEY_MEMBER** (p. 240) fields. If you do not set a value of the member explicitly (see, for example, **DDS_DynamicData_set_long** (p. 394)), 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. 241)

DDS_TYPECODE_NONKEY_MEMBER (p. 240)

DDS_TypeCode_add_member (p. 280)

DDS_TypeCode_add_member_ex (p. 282)

DDS_TypeCode_is_member_key (p. 261)

DDS_TypeCode_is_member_required (p. 262)

DDS_StructMember::is_key (p. 2008)

DDS_ValueMember::is_key (p. 2089)

5.11.4.12 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. 313) facility, all **DDS_DynamicData** (p. 1768) samples of the type will contain a value for all **DDS_TYPECODE_NONKEY_REQUIRED_MEMBER** (p. 241) fields. If you do not set a value of the member explicitly (see, for example, **DDS_DynamicData_set_long** (p. 394)), 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. 240)

DDS_TYPECODE_NONKEY_MEMBER (p. 240)

DDS_TYPECODE_KEY_MEMBER (p. 240)

DDS_TypeCode_add_member (p. 280)

DDS_TypeCode_add_member_ex (p. 282)

DDS_TypeCode_is_member_key (p. 261)

DDS_TypeCode_is_member_required (p. 262)

DDS_StructMember::is_key (p. 2008)

DDS_ValueMember::is_key (p. 2089)

5.11.4.13 DDS_TypeCode_PrintFormat_INITIALIZER

```
#define DDS_TypeCode_PrintFormat_INITIALIZER
```

Static initializer for **DDS_TypeCodePrintFormatProperty** (p. 2073).

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

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

5.11.5 Typedef Documentation

5.11.5.1 DDS_ValueModifier

```
typedef short DDS_ValueModifier
```

Modifier type for a value type.

See also

DDS_VM_NONE (p. 238)

DDS_VM_CUSTOM (p. 238)

DDS_VM_ABSTRACT (p. 239)

DDS_VM_TRUNCATABLE (p. 239)

5.11.5.2 DDS_Visibility

```
typedef short DDS_Visibility
```

Type to indicate the visibility of a value type member.

See also

DDS_PRIVATE_MEMBER (p. 239)

DDS_PUBLIC_MEMBER (p. 240)

5.11.6 Enumeration Type Documentation

5.11.6.1 DDS_TypeCodePrintFormatKind

```
enum DDS_TypeCodePrintFormatKind
```

The different formats to print a **DDS_TypeCode** (p. 2071).

Enumerator

DDS_TYPE_CODE_PRINT_KIND_IDL	Indicates that a DDS_TypeCode (p. 2071) is to be printed in IDL format.
DDS_TYPE_CODE_PRINT_KIND_XML	Indicates that a DDS_TypeCode (p. 2071) is to be printed in XML format.

5.11.6.2 DDS_TCKind

```
enum DDS_TCKind
```

Enumeration type for **DDS_TypeCode** (p. 2071) 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.

5.11.6.3 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. 243) • 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. 243) • DDS_TK_STRING (p. 243) • DDS_TK_WSTRING (p. 243)

5.11.7 Function Documentation

5.11.7.1 DDS_TypeCode_kind()

```
DDS_TCKind DDS_TypeCode_kind (
    const DDS_TypeCode * self,
    DDS_StatusCode_t * ex )
```

Gets the **DDS_TCKind** (p. 243) value of a type code.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Parameters

<i>ex</i>	<< out >> (p. 1013) Parameter for error indications. The values that it can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220)

Retrieves the kind of this **DDS_TypeCode** (p. 2071) object. The kind of a type code determines which **DDS_TypeCode** (p. 2071) functions may legally be invoked on it.

MT Safety:

SAFE.

Returns

The type code kind.

5.11.7.2 **DDS_TypeCode_extensibility_kind()**

```

DDS_ExtensibilityKind DDS_TypeCode_extensibility_kind (
    const DDS_TypeCode * self,
    DDS_ExceptionCode_t * ex )

```

Gets the **DDS_ExtensibilityKind** (p. 243) value of a type code.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>ex</i>	<< out >> (p. 1013) Parameter for error indications. The values that it can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220)

Retrieves the extensibility kind of this **DDS_TypeCode** (p. 2071) 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. 243), **DDS_TK_UNION** (p. 243), **DDS_TK_VALUE** (p. 243), and **DDS_TK_ENUM** (p. 243) 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. 2071) API, the extensibility can be provided as a parameter of the following APIs:

- **DDS_TypeCodeFactory_create_struct_tc_ex** (p. 292)
- **DDS_TypeCodeFactory_create_value_tc_ex** (p. 293)
- **DDS_TypeCodeFactory_create_union_tc_ex** (p. 295)
- **DDS_TypeCodeFactory_create_enum_tc_ex** (p. 296)

See also

DDS_ExtensibilityKind (p. 243)

MT Safety:

SAFE.

Returns

The type code extensibility kind.

5.11.7.3 DDS_TypeCode_equal()

```
DDS_Boolean DDS_TypeCode_equal (
    const DDS_TypeCode * self,
    const DDS_TypeCode * tc,
    DDS_ExceptionCode_t * ex )
```

Compares two **DDS_TypeCode** (p. 2071) objects for equality.

MT Safety:

SAFE.

For equality and assignability purposes, **DDS_TK_STRUCT** (p. 243) and **DDS_TK_VALUE** (p. 243) are considered equivalent.

The **DDS_TypeCode** (p. 2071) of structs inheriting from other structs has a **DDS_TK_VALUE** (p. 243) kind.

For example:

```
struct MyStruct: MyBaseStruct {
    long member_1;
};
```

The code generation for the previous type will generate a **DDS_TypeCode** (p. 2071) with **DDS_TK_VALUE** (p. 243) kind.

Additionally, for equality and assignability purposes, **DDS_TK_OCTET** (p. 243), **DDS_TK_INT8** (p. 243), and **DDS_TK_UINT8** (p. 243) are considered equivalent.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>tc</i>	<< <i>in</i> >> (p. 1013) Type code that will be compared with this DDS_TypeCode (p. 2071).
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220)

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the type codes are equal. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201).

5.11.7.4 DDS_TypeCode_name()

```
const char * DDS_TypeCode_name (
    const DDS_TypeCode * self,
    DDS_ExceptionCode_t * ex )
```

Retrieves the simple name identifying this **DDS_TypeCode** (p. 2071) object within its enclosing scope.

Precondition

self kind is **DDS_TK_STRUCT** (p. 243), **DDS_TK_UNION** (p. 243), **DDS_TK_ENUM** (p. 243), **DDS_TK_VALUE** (p. 243), or **DDS_TK_ALIAS** (p. 243).

MT Safety:

SAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220)

Returns

Name of the type code if no errors.

5.11.7.5 DDS_TypeCode_default_annotation()

```
const DDS_TypeCode_Annotation * DDS_TypeCode_default_annotation (
    const DDS_TypeCode * self,
    DDS_ExceptionCode_t * ex )
```

[DEPRECATED] Given a **DDS_TypeCode** (p. 2071) object, retrieves its **[default]** annotation value as a **DDS_TypeCode_Annotation**

DEPRECATED: use **DDS_TypeCode_default_value** (p. 250)

5.11.7.6 DDS_TypeCode_member_default_annotation()

```
const DDS_TypeCode_Annotation * DDS_TypeCode_member_default_annotation (
    const DDS_TypeCode * self,
    DDS_UnsignedLong index,
    DDS_ExceptionCode_t * ex )
```

[DEPRECATED] Returns the **[default]** annotation of a type code member identified by the given index.

DEPRECATED: use **DDS_TypeCode_member_default_value** (p. 252)

5.11.7.7 DDS_TypeCode_min_annotation()

```
const DDS_TypeCode_Annotation * DDS_TypeCode_min_annotation (
    const DDS_TypeCode * self,
    DDS_ExceptionCode_t * ex )
```

[DEPRECATED] Given a **DDS_TypeCode** (p. 2071) object, retrieves its @min annotation value as a **DDS_TypeCode↵_Annotation**

DEPRECATED: use **DDS_TypeCode_min_value** (p. 251)

5.11.7.8 DDS_TypeCode_max_annotation()

```
const DDS_TypeCode_Annotation * DDS_TypeCode_max_annotation (
    const DDS_TypeCode * self,
    DDS_ExceptionCode_t * ex )
```

[DEPRECATED] Given a **DDS_TypeCode** (p. 2071) object, retrieves its @max annotation value as a **DDS_TypeCode↵_Annotation**

DEPRECATED: use **DDS_TypeCode_max_value** (p. 252)

5.11.7.9 DDS_TypeCode_default_value()

```
const DDS_AnnotationParameterValue * DDS_TypeCode_default_value (
    const DDS_TypeCode * self,
    DDS_ExceptionCode_t * ex )
```

Returns the default annotation for this **DDS_TypeCode** (p. 2071).

Precondition

self kind is **DDS_TK_ENUM** (p. 243), or **DDS_TK_ALIAS** (p. 243).

MT Safety:

SAFE

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220)

Returns

the default annotation for this **DDS_TypeCode** (p. 2071)

5.11.7.10 DDS_TypeCode_min_value()

```
const DDS_AnnotationParameterValue * DDS_TypeCode_min_value (
    const DDS_TypeCode * self,
    DDS_ExceptionCode_t * ex )
```

Returns the min annotation for this **DDS_TypeCode** (p. 2071).

Precondition

self kind is **DDS_TK_ENUM** (p. 243), or **DDS_TK_ALIAS** (p. 243) of a numerical kind.

MT Safety:

SAFE

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220)

Returns

the min annotation for this **DDS_TypeCode** (p. 2071)

5.11.7.11 DDS_TypeCode_max_value()

```
const DDS_AnnotationParameterValue * DDS_TypeCode_max_value (
    const DDS_TypeCode * self,
    DDS_ExceptionCode_t * ex )
```

Returns the max annotation for this **DDS_TypeCode** (p. 2071).

Precondition

self kind is **DDS_TK_ENUM** (p. 243), or **DDS_TK_ALIAS** (p. 243) of a numerical kind.

MT Safety:

SAFE

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220)

Returns

the max annotation for this **DDS_TypeCode** (p. 2071)

5.11.7.12 DDS_TypeCode_member_default_value()

```
const DDS_AnnotationParameterValue * DDS_TypeCode_member_default_value (
    const DDS_TypeCode * self,
```

```

    DDS_UnsignedLong index,
    DDS_ExceptionCode_t * ex )

```

Returns the member default annotation for this **DDS_TypeCode** (p. 2071).

Precondition

self kind is **DDS_TK_STRUCT** (p. 243), or **DDS_TK_UNION** (p. 243).

MT Safety:

SAFE

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>index</i>	index of the target member
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220) • DDS_BOUNDS_USER_EXCEPTION_CODE (p. 1220)

Returns

the member default annotation for this **DDS_TypeCode** (p. 2071)

5.11.7.13 DDS_TypeCode_member_min_value()

```

const DDS_AnnotationParameterValue * DDS_TypeCode_member_min_value (
    const DDS_TypeCode * self,
    DDS_UnsignedLong index,
    DDS_ExceptionCode_t * ex )

```

Returns the member min annotation for this **DDS_TypeCode** (p. 2071).

Precondition

self kind is **DDS_TK_STRUCT** (p. 243), or **DDS_TK_UNION** (p. 243).
 member pointed to by index of a numerical kind.

MT Safety:

SAFE

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>index</i>	index of the target member
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220) • DDS_BOUNDS_USER_EXCEPTION_CODE (p. 1220)

Returns

the member min annotation for this **DDS_TypeCode** (p. 2071)

5.11.7.14 DDS_TypeCode_member_max_value()

```
const DDS_AnnotationParameterValue * DDS_TypeCode_member_max_value (
    const DDS_TypeCode * self,
    DDS_UnsignedLong index,
    DDS_ExceptionCode_t * ex )
```

Returns the member max annotation for this **DDS_TypeCode** (p. 2071).

Precondition

self kind is **DDS_TK_STRUCT** (p. 243), or **DDS_TK_UNION** (p. 243).
 member pointed to by index of a numerical kind.

MT Safety:

SAFE

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>index</i>	index of the target member
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220) • DDS_BOUNDS_USER_EXCEPTION_CODE (p. 1220)

Returns

the member max annotation for this **DDS_TypeCode** (p. 2071)

5.11.7.15 DDS_TypeCode_member_count()

```
DDS_UnsignedLong DDS_TypeCode_member_count (
    const DDS_TypeCode * self,
    DDS_ExceptionCode_t * ex )
```

Returns the number of members of the type code.

The function member_count can be invoked on structure, union, and enumeration **DDS_TypeCode** (p. 2071) objects.

Precondition

self kind is **DDS_TK_STRUCT** (p. 243), **DDS_TK_UNION** (p. 243), **DDS_TK_ENUM** (p. 243) or **DDS_TK_VALUE** (p. 243).

MT Safety:

SAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220)

Returns

The number of members constituting the type described by this **DDS_TypeCode** (p. 2071) object if no errors.

5.11.7.16 DDS_TypeCode_member_name()

```
const char * DDS_TypeCode_member_name (
    const   DDS_TypeCode * self,
    DDS_UnsignedLong index,
    DDS_ExceptionCode_t * ex )
```

Returns the name of a type code member identified by the given index.

The function member_name can be invoked on structure, union, and enumeration **DDS_TypeCode** (p. 2071) objects.

Precondition

self kind is **DDS_TK_STRUCT** (p. 243), **DDS_TK_UNION** (p. 243), **DDS_TK_ENUM** (p. 243) or **DDS_TK_VALUE** (p. 243).

The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>index</i>	<< <i>in</i> >> (p. 1013) Member index in the interval [0,(member count-1)].
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220) • DDS_BOUNDS_USER_EXCEPTION_CODE (p. 1220)

Returns

Name of the member if no errors.

5.11.7.17 DDS_TypeCode_find_member_by_name()

```
DDS_UnsignedLong DDS_TypeCode_find_member_by_name (
    const DDS_TypeCode * self,
    const char * name,
    DDS_ExceptionCode_t * ex )
```

Get the index of the member of the given name.

This method is applicable to **DDS_TypeCode** (p. 2071) objects representing structs (**DDS_TK_STRUCT** (p. 243)) and union (**DDS_TK_UNION** (p. 243)) types.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>name</i>	<< <i>in</i> >> (p. 1013) The member name.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220)

Returns

The index of the member of the given name or **DDS_TYPECODE_INDEX_INVALID** (p. 238) if the member is not found.

MT Safety:

SAFE.

5.11.7.18 DDS_TypeCode_member_type()

```
DDS_TypeCode * DDS_TypeCode_member_type (
    const DDS_TypeCode * self,
    DDS_UnsignedLong index,
    DDS_ExceptionCode_t * ex )
```

Retrieves the **DDS_TypeCode** (p. 2071) object describing the type of the member identified by the given index.

The function `member_type` can be invoked on structure and union type codes.

Precondition

self kind is **DDS_TK_STRUCT** (p. 243), **DDS_TK_UNION** (p. 243) or **DDS_TK_VALUE** (p. 243).
 The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>index</i>	<< <i>in</i> >> (p. 1013) Member index in the interval [0,(member count-1)].
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220) • DDS_BOUNDS_USER_EXCEPTION_CODE (p. 1220)

Returns

The **DDS_TypeCode** (p. 2071) object describing the member at the given index if no errors.

5.11.7.19 DDS_TypeCode_member_label_count()

```
DDS_UnsignedLong DDS_TypeCode_member_label_count (
    const DDS_TypeCode * self,
    DDS_UnsignedLong index,
    DDS_StatusCode_t * ex )
```

Returns the number of labels associated to the index-th union member.

The function can be invoked on union **DDS_TypeCode** (p. 2071) objects.

This function is an RTI Connex extension to the CORBA Type Code Specification.

Precondition

self kind is **DDS_TK_UNION** (p. 243).
 The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>index</i>	<< <i>in</i> >> (p. 1013) Member index in the interval [0,(member count-1)].
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220) • DDS_BOUNDS_USER_EXCEPTION_CODE (p. 1220)

Returns

Number of labels if no errors.

5.11.7.20 DDS_TypeCode_member_label()

```
DDS_Long DDS_TypeCode_member_label (
    const DDS_TypeCode * self,
    DDS_UnsignedLong member_index,
    DDS_UnsignedLong label_index,
    DDS_ExceptionCode_t * ex )
```

Return the label_index-th label associated to the member_index-th member.

This function has been modified for RTI Connex from the CORBA Type code Specification.

Example:

case 1: Label index 0

case 2: Label index 1

short short_member;

The function can be invoked on union **DDS_TypeCode** (p. 2071) objects.

Precondition

self kind is **DDS_TK_UNION** (p. 243).

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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_index</i>	<< <i>in</i> >> (p. 1013) Member index.
<i>label_index</i>	<< <i>in</i> >> (p. 1013) Label index.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220) • DDS_BOUNDS_USER_EXCEPTION_CODE (p. 1220)

Returns

The evaluated value of the label if no errors.

5.11.7.21 DDS_TypeCode_member_ordinal()

```

DDS_Long DDS_TypeCode_member_ordinal (
    const DDS_TypeCode * self,
    DDS_UnsignedLong index,
    DDS_ExceptionCode_t * ex )

```

Returns the ordinal that corresponds to the index-th enum value.

The function can be invoked on enum **DDS_TypeCode** (p. 2071) objects.

This function is an RTI Connext extension to the CORBA Type Code Specification.

Precondition

self kind is **DDS_TK_ENUM** (p. 243).
 Member index in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>index</i>	<< <i>in</i> >> (p. 1013) Member index in the interval [0,(member count-1)].
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220) • DDS_BOUNDS_USER_EXCEPTION_CODE (p. 1220)

Returns

Ordinal that corresponds to the index-th enumerator if no errors.

5.11.7.22 DDS_TypeCode_is_member_key()

```

DDS_Boolean DDS_TypeCode_is_member_key (
    const DDS_TypeCode * self,
    DDS_UnsignedLong index,
    DDS_ExceptionCode_t * ex )

```

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. 243) or **DDS_TK_VALUE** (p. 243).
 The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>index</i>	<< <i>in</i> >> (p. 1013) Member index in the interval [0,(member count-1)].

Parameters

<i>ex</i>	<<<i>out</i>>> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220) • DDS_BOUNDS_USER_EXCEPTION_CODE (p. 1220)

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the member is a key. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201).

5.11.7.23 **DDS_TypeCode_is_member_required()**

```
DDS_Boolean DDS_TypeCode_is_member_required (
    const DDS_TypeCode * self,
    DDS_UnsignedLong index,
    DDS_StatusCode_t * ex )
```

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. 240)

DDS_TYPECODE_NONKEY_REQUIRED_MEMBER (p. 241)

MT Safety:

SAFE.

5.11.7.24 DDS_TypeCode_is_member_pointer()

```
DDS_Boolean DDS_TypeCode_is_member_pointer (
    const DDS_TypeCode * self,
    DDS_UnsignedLong index,
    DDS_ExceptionCode_t * ex )
```

Function that tells if a member is a pointer or not.

The function `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. 243), **DDS_TK_UNION** (p. 243) or **DDS_TK_VALUE** (p. 243).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>index</i>	<< <i>in</i> >> (p. 1013) Index of the member for which type information is begin requested.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220) • DDS_BOUNDS_USER_EXCEPTION_CODE (p. 1220)

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the member is a pointer. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201).

5.11.7.25 DDS_TypeCode_is_member_bitfield()

```
DDS_Boolean DDS_TypeCode_is_member_bitfield (
    const DDS_TypeCode * self,
```

```

    DDS_UnsignedLong index,
    DDS_ExceptionCode_t * ex )

```

Function that tells if a member is a bitfield or not.

The function 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. 243) or **DDS_TK_VALUE** (p. 243).
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>index</i>	<< <i>in</i> >> (p. 1013) Member index in the interval [0,(member count-1)].
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220) • DDS_BOUNDS_USER_EXCEPTION_CODE (p. 1220)

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the member is a bitfield. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201).

5.11.7.26 DDS_TypeCode_member_bitfield_bits()

```

DDS_Short DDS_TypeCode_member_bitfield_bits (
    const DDS_TypeCode * self,
    DDS_UnsignedLong index,
    DDS_ExceptionCode_t * ex )

```

Returns the number of bits of a bitfield member.

The function 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. 243) or **DDS_TK_VALUE** (p. 243).
 The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>index</i>	<< <i>in</i> >> (p. 1013) Member index in the interval [0,(member count-1)].
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220) • DDS_BOUNDS_USER_EXCEPTION_CODE (p. 1220)

Returns

The number of bits of the bitfield or **DDS_TYPECODE_NOT_BITFIELD** (p. 238) if the member is not a bitfield.

5.11.7.27 DDS_TypeCode_member_visibility()

```

DDS_Visibility DDS_TypeCode_member_visibility (
    const DDS_TypeCode * self,
    DDS_UnsignedLong index,
    DDS_ExceptionCode_t * ex )
  
```

Returns the constant that indicates the visibility of the index-th member.

Precondition

self kind is **DDS_TK_VALUE** (p. 243) or **DDS_TK_STRUCT** (p. 243). The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

For **DDS_TK_STRUCT** (p. 243), this function always returns **DDS_PUBLIC_MEMBER** (p. 240).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>index</i>	<< <i>in</i> >> (p. 1013) Member index in the interval [0,(member count-1)].
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220) • DDS_BOUNDS_USER_EXCEPTION_CODE (p. 1220)

Returns

One of the following constants: **DDS_PRIVATE_MEMBER** (p. 239) or **DDS_PUBLIC_MEMBER** (p. 240).

5.11.7.28 DDS_TypeCode_discriminator_type()

```
DDS_TypeCode * DDS_TypeCode_discriminator_type (
    const DDS_TypeCode * self,
    DDS_ExceptionCode_t * ex )
```

Returns the discriminator type code.

The function discriminator_type can be invoked only on union **DDS_TypeCode** (p. 2071) objects.

Precondition

self kind is **DDS_TK_UNION** (p. 243).

MT Safety:

SAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220)

Returns

DDS_TypeCode (p. 2071) object describing the discriminator of the union type if no errors.

5.11.7.29 DDS_TypeCode_length()

```
DDS_UnsignedLong DDS_TypeCode_length (
    const DDS_TypeCode * self,
    DDS_ExceptionCode_t * ex )
```

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. 243), **DDS_TK_SEQUENCE** (p. 243), **DDS_TK_STRING** (p. 243) or **DDS_TK_WSTRING** (p. 243).

MT Safety:

SAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220)

Returns

The bound for strings and sequences, or the number of elements for arrays if no errors.

5.11.7.30 DDS_TypeCode_array_dimension_count()

```
DDS_UnsignedLong DDS_TypeCode_array_dimension_count (
    const DDS_TypeCode * self,
    DDS_ExceptionCode_t * ex )
```

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. 243).

MT Safety:

SAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220)

Returns

Number of dimensions if no errors.

5.11.7.31 DDS_TypeCode_array_dimension()

```
DDS_UnsignedLong DDS_TypeCode_array_dimension (
    const DDS_TypeCode * self,
```

```

    DDS_UnsignedLong index,
    DDS_ExceptionCode_t * ex )

```

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. 243).
 Dimension index in the interval [0,(dimensions count-1)].

MT Safety:

SAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>index</i>	<< <i>in</i> >> (p. 1013) Dimension index in the interval [0,(dimensions count-1)].
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220) • DDS_BOUNDS_USER_EXCEPTION_CODE (p. 1220)

Returns

Requested dimension if no errors.

5.11.7.32 DDS_TypeCode_element_count()

```

    DDS_UnsignedLong DDS_TypeCode_element_count (
        const DDS_TypeCode * self,
        DDS_ExceptionCode_t * ex )

```

The number of elements in an array.

This operation isn't relevant for other kinds of types.

MT Safety:

SAFE.

5.11.7.33 DDS_TypeCode_content_type()

```
DDS_TypeCode * DDS_TypeCode_content_type (
    const DDS_TypeCode * self,
    DDS_ExceptionCode_t * ex )
```

Returns the **DDS_TypeCode** (p. 2071) object representing the type for the members of the object described by this **DDS_TypeCode** (p. 2071) 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. 243), **DDS_TK_SEQUENCE** (p. 243) or **DDS_TK_ALIAS** (p. 243).

MT Safety:

SAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220)

Returns

A **DDS_TypeCode** (p. 2071) object representing the element type for sequences and arrays, and the original type for aliases.

5.11.7.34 DDS_TypeCode_is_alias_pointer()

```
DDS_Boolean DDS_TypeCode_is_alias_pointer (
    const DDS_TypeCode * self,
    DDS_ExceptionCode_t * ex )
```

Function that tells if an alias is a pointer or not.

This function is an RTI Connex extension to the CORBA Type Code Specification.

Precondition

self kind is **DDS_TK_ALIAS** (p. 243).

MT Safety:

SAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220)

Returns

DDS_BOOLEAN_TRUE (p. 1201) if an alias is a pointer to the aliased type. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201).

5.11.7.35 DDS_TypeCode_default_index()

```
DDS_Long DDS_TypeCode_default_index (
    const DDS_TypeCode * self,
    DDS_ExceptionCode_t * ex )
```

Returns the index of the default member, or -1 if there is no default member.

The function `default_index` can be invoked only on union **DDS_TypeCode** (p. 2071) objects.

Precondition

self kind is **DDS_TK_UNION** (p. 243)

MT Safety:

SAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220)

Returns

The index of the default member, or -1 if there is no default member.

5.11.7.36 DDS_TypeCode_concrete_base_type()

```
DDS_TypeCode * DDS_TypeCode_concrete_base_type (
    const DDS_TypeCode * self,
    DDS_ExceptionCode_t * ex )
```

Returns the **DDS_TypeCode** (p. 2071) that describes the concrete base type of the value type that this **DDS_TypeCode** (p. 2071) object describes.

Precondition

self kind is **DDS_TK_VALUE** (p. 243) or **DDS_TK_STRUCT** (p. 243).

MT Safety:

SAFE.

For **DDS_TK_STRUCT** (p. 243), this function always returns **DDS_TK_NULL** (p. 243).

The **DDS_TypeCode** (p. 2071) of structs inheriting from other structs has a **DDS_TK_VALUE** (p. 243) kind.

For example:

```
struct MyStruct: MyBaseStruct {  
  
    long member_1;  
  
};
```

The code generation for the previous type will generate a **DDS_TypeCode** (p. 2071) with **DDS_TK_VALUE** (p. 243) kind.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220)

Returns

DDS_TypeCode (p. 2071) that describes the concrete base type or NULL if there is no a concrete base type.

5.11.7.37 DDS_TypeCode_type_modifier()

```

DDS_ValueModifier DDS_TypeCode_type_modifier (
    const DDS_TypeCode * self,
    DDS_ExceptionCode_t * ex )

```

Returns a constant indicating the modifier of the value type that this **DDS_TypeCode** (p. 2071) object describes.

Precondition

self kind is **DDS_TK_VALUE** (p. 243) or **DDS_TK_STRUCT** (p. 243).

For **DDS_TK_STRUCT** (p. 243), this function always returns **DDS_VM_NONE** (p. 238).

MT Safety:

SAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220)

Returns

One of the following type modifiers: **DDS_VM_NONE** (p. 238), **DDS_VM_ABSTRACT** (p. 239), **DDS_VM_↵CUSTOM** (p. 238) or **DDS_VM_TRUNCATABLE** (p. 239).

5.11.7.38 DDS_TypeCode_find_member_by_id()

```
DDS_UnsignedLong DDS_TypeCode_find_member_by_id (
    const DDS_TypeCode * self,
    DDS_Long id,
    DDS_ExceptionCode_t * ex )
```

Get the index of the member of the given ID.

This method is applicable to **DDS_TypeCode** (p. 2071) objects representing structs (**DDS_TK_STRUCT** (p. 243)) and union (**DDS_TK_UNION** (p. 243)) types.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>id</i>	<< <i>in</i> >> (p. 1013) The member ID.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220)

Returns

The index of the member of the given ID or **DDS_TYPECODE_INDEX_INVALID** (p. 238) if the member is not found.

MT Safety:

SAFE.

5.11.7.39 DDS_TypeCode_member_id()

```
DDS_Long DDS_TypeCode_member_id (
    const DDS_TypeCode * self,
    DDS_UnsignedLong index,
    DDS_ExceptionCode_t * ex )
```

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 function can be invoked on aggregation **DDS_TypeCode** (p. 2071) 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. 243), **DDS_TK_VALUE** (p. 243), or **DDS_TK_UNION** (p. 243)
Member index in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>index</i>	<< <i>in</i> >> (p. 1013) Member index in the interval [0,(member count-1)].

Parameters

<i>ex</i>	<<out>> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220) • DDS_BOUNDS_USER_EXCEPTION_CODE (p. 1220)

Returns

ID of the member if no errors.

5.11.7.40 **DDS_TypeCode_get_type_object_serialized_size()**

```

DDS_UnsignedLong DDS_TypeCode_get_type_object_serialized_size (
    const DDS_TypeCode * self,
    DDS_StatusCode_t * ex )

```

Gets the serialized size of the TypeObject created from this **DDS_TypeCode** (p. 2071).

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>ex</i>	<<out>> (p. 1013) Parameter for error indications. The values that it can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_SYSTEM_EXCEPTION_CODE (p. 1219)

The default buffer size used for storing a TypeObject is 8192 bytes. For a large TypeObject, this API can be used to determine the size that needs to be set in **DDS_DomainParticipantResourceLimitsQosPolicy::type_object_max_↵_serialized_length** (p. 1749)

MT Safety:

SAFE.

Returns

The TypeObject serialized size.

5.11.7.41 DDS_TypeCode_add_member_to_enum()

```
DDS_UnsignedLong DDS_TypeCode_add_member_to_enum (
    DDS_TypeCode * self,
    const char * name,
    DDS_Long ordinal,
    DDS_ExceptionCode_t * ex )
```

Add a new enumerated constant to this enum **DDS_TypeCode** (p. 2071).

This method is applicable to **DDS_TypeCode** (p. 2071) objects representing enumerations (**DDS_TK_ENUM** (p. 243)). To add a field to a structured type, see **DDS_TypeCode_add_member_to_enum** (p. 278).

Modifying a **DDS_TypeCode** (p. 2071) – such as by adding a member – is important if you are using the **Dynamic Data** (p. 313) APIs.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>name</i>	<< <i>in</i> >> (p. 1013) The name of the new member. This string must be unique within this type and must not be NULL.
<i>ordinal</i>	<< <i>in</i> >> (p. 1013) The relative order of the new member in this enum or a custom integer value. The value must be unique within the type.
<i>ex</i>	<< <i>out</i> >> (p. 1013) If this method fails, this argument will contain information about the failure. Possible values include: • DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220) • DDS_BAD_MEMBER_NAME_USER_EXCEPTION_CODE (p. 1220) • DDS_BAD_MEMBER_ID_USER_EXCEPTION_CODE (p. 1220)

Returns

The zero-based index of the new member relative to any other members that previously existed.

See also

DDS_TypeCode_add_member (p. 280)

DDS_TypeCode_add_member_ex (p. 282)

DDS_TypeCodeFactory (p. 2071)

5.11.7.42 DDS_TypeCode_add_member_to_union()

```
DDS_UnsignedLong DDS_TypeCode_add_member_to_union (
    DDS_TypeCode * self,
    const char * name,
    DDS_Long id,
    const struct 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. 2071).

This method is applicable to **DDS_TypeCode** (p. 2071) objects representing unions (**DDS_TK_UNION** (p. 243)).

Modifying a **DDS_TypeCode** (p. 2071) – such as by adding a member – is important if you are using the **Dynamic Data** (p. 313) APIs.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>name</i>	<< <i>in</i> >> (p. 1013) The name of the new member.
<i>id</i>	<< <i>in</i> >> (p. 1013) The member ID. For automatic assignment of the member ID specify DDS_TYPECODE_MEMBER_ID_INVALID (p. 237).
<i>labels</i>	<< <i>in</i> >> (p. 1013) A sequence of labels or case values associated with the member .
<i>tc</i>	<< <i>in</i> >> (p. 1013) The type of the new member. You can get or create this DDS_TypeCode (p. 2071) with the DDS_TypeCodeFactory (p. 2071).
<i>is_pointer</i>	<< <i>in</i> >> (p. 1013) Whether the data member, in its deserialized form, should be stored by pointer as opposed to by value.
<i>ex</i>	<< <i>out</i> >> (p. 1013) If this method fails, this argument will contain information about the failure.

Possible values include:

- **DDS_BADKIND_USER_EXCEPTION_CODE** (p. 1220)
- **DDS_BAD_MEMBER_NAME_USER_EXCEPTION_CODE** (p. 1220)

- **DDS_BAD_MEMBER_ID_USER_EXCEPTION_CODE** (p. 1220)

Returns

The zero-based index of the new member relative to any other members that previously existed.

See also

DDS_TypeCode_add_member (p. 280)
DDS_TypeCode_add_member_ex (p. 282)
DDS_TypeCode_add_member_to_enum (p. 278)
DDS_TypeCode_add_member_to_union (p. 279)
DDS_TypeCodeFactory (p. 2071)
DDS_TYPECODE_NONKEY_MEMBER (p. 240)
DDS_TYPECODE_KEY_MEMBER (p. 240)
DDS_TYPECODE_NONKEY_REQUIRED_MEMBER (p. 241)

5.11.7.43 DDS_TypeCode_add_member()

```

DDS_UnsignedLong DDS_TypeCode_add_member (
    DDS_TypeCode * self,
    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. 2071).

This method is applicable to **DDS_TypeCode** (p. 2071) objects representing structures (**DDS_TK_STRUCT** (p. 243)), value types (**DDS_TK_VALUE** (p. 243)), and unions (**DDS_TK_UNION** (p. 243)). To add a constant to an enumeration, see **DDS_TypeCode_add_member_to_enum** (p. 278).

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. 290).

The ability to modify a **DDS_TypeCode** (p. 2071) – such as by adding a member – is important if you are using the **Dynamic Data** (p. 313) 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:
DDS_TypeCode_add_member(
    myTypeCode,
    "myFieldName",
    DDS_TYPECODE_MEMBER_ID_INVALID,
  
```



```

DDS_TypeCodeFactory_get_primitive_tc(DDS_TheTypeCodeFactory, DDS_TK_LONG),

// New field is a required non-key member:
DDS_TYPECODE_NONKEY_REQUIRED_MEMBER);

// Sequence of 10 or fewer integers:
DDS_TypeCode_add_member(
    myTypeCode,
    "myFieldName",
    DDS_TYPECODE_MEMBER_ID_INVALID,
    DDS_TypeCodeFactory_create_sequence_tc(
        DDS_TheTypeCodeFactory,
        10,
        DDS_TypeCodeFactory_get_primitive_tc(DDS_TheTypeCodeFactory, DDS_TK_LONG)),
    // New field is a required non-key member:
    DDS_TYPECODE_NONKEY_REQUIRED_MEMBER);

```

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>name</i>	<< <i>in</i> >> (p. 1013) The name of the new member.
<i>id</i>	<< <i>in</i> >> (p. 1013) Member ID or case value. - For DDS_TK_STRUCT (p. 243), and DDS_TK_VALUE (p. 243), this parameter is used to provide the member ID. For automatic assignment of the member ID specify DDS_TYPECODE_MEMBER_ID_INVALID (p. 237). - For DDS_TK_UNION (p. 243), 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. 279).
<i>tc</i>	<< <i>in</i> >> (p. 1013) The type of the new member. You can get or create this DDS_TypeCode (p. 2071) with the DDS_TypeCodeFactory (p. 2071).
<i>member_flags</i>	<< <i>in</i> >> (p. 1013) Indicates whether the member is part of the key and whether it is required.
<i>ex</i>	<< <i>out</i> >> (p. 1013) If this method fails, this argument will contain information about the failure. Possible values include: DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220) DDS_BAD_MEMBER_NAME_USER_EXCEPTION_CODE (p. 1220) DDS_BAD_MEMBER_ID_USER_EXCEPTION_CODE (p. 1220)

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. 282)
DDS_TypeCode_add_member_to_enum (p. 278)
DDS_TypeCodeFactory (p. 2071)
DDS_TYPECODE_NONKEY_MEMBER (p. 240)
DDS_TYPECODE_KEY_MEMBER (p. 240)
DDS_TYPECODE_NONKEY_REQUIRED_MEMBER (p. 241)

5.11.7.44 DDS_TypeCode_add_member_ex()

```

DDS_UnsignedLong DDS_TypeCode_add_member_ex (
    DDS_TypeCode * self,
    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_ExceptionCode_t * ex )
  
```

Add a new member to this **DDS_TypeCode** (p. 2071).

This method is applicable to **DDS_TypeCode** (p. 2071) objects representing structures (**DDS_TK_STRUCT** (p. 243)), value types (**DDS_TK_VALUE** (p. 243)), and unions (**DDS_TK_UNION** (p. 243)). To add a constant to an enumeration, see **DDS_TypeCode_add_member_to_enum** (p. 278).

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. 290).

The ability to modify a **DDS_TypeCode** (p. 2071) – such as by adding a member – is important if you are using the **Dynamic Data** (p. 313) APIs.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>name</i>	<< <i>in</i> >> (p. 1013) The name of the new member.

Parameters

<i>id</i>	<<<i>in</i>>> (p. 1013) Member ID or case value. - For DDS_TK_STRUCT (p. 243) and DDS_TK_VALUE (p. 243), this parameter is used to provide the member ID. - For DDS_TK_STRUCT (p. 243) and DDS_TK_VALUE (p. 243) only: For automatic assignment of the member ID, specify DDS_TYPECODE_MEMBER_ID_INVALID (p. 237). - For DDS_TK_UNION (p. 243), 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. 279).
<i>tc</i>	<<<i>in</i>>> (p. 1013) The type of the new member. You can get or create this DDS_TypeCode (p. 2071) with the DDS_TypeCodeFactory (p. 2071).
<i>member_flags</i>	<<<i>in</i>>> (p. 1013) Indicates whether the member is part of the key and whether it is required.
<i>visibility</i>	<<<i>in</i>>> (p. 1013) Whether the new member is public or private. Non-public members are only relevant for types of kind DDS_TK_VALUE (p. 243). Possible values include: DDS_PRIVATE_MEMBER (p. 239) DDS_PUBLIC_MEMBER (p. 240)
<i>is_pointer</i>	<<<i>in</i>>> (p. 1013) Whether the data member, in its deserialized form, should be stored by pointer as opposed to by value.
<i>bits</i>	<<<i>in</i>>> (p. 1013) The number of bits, if this new member is a bit field, or DDS_TYPECODE_NOT_BITFIELD (p. 238).
<i>ex</i>	<<<i>out</i>>> (p. 1013) If this method fails, this argument will contain information about the failure. Possible values include: DDS_BADKIND_USER_EXCEPTION_CODE (p. 1220) DDS_BAD_MEMBER_NAME_USER_EXCEPTION_CODE (p. 1220) DDS_BAD_MEMBER_ID_USER_EXCEPTION_CODE (p. 1220)

Returns

The zero-based index of the new member relative to any other members that previously existed.

See also

DDS_TypeCode_add_member (p. 280)
DDS_TypeCode_add_member_to_enum (p. 278)
DDS_TypeCode_add_member_to_union (p. 279)
DDS_TypeCodeFactory (p. 2071)
DDS_TYPECODE_NONKEY_MEMBER (p. 240)
DDS_TYPECODE_KEY_MEMBER (p. 240)
DDS_TYPECODE_NONKEY_REQUIRED_MEMBER (p. 241)

5.11.7.45 DDS_TypeCode_print_IDL()

```
void DDS_TypeCode_print_IDL (
    const DDS_TypeCode * self,
    DDS_UnsignedLong indent,
    DDS_ExceptionCode_t * ex )
```

Prints a **DDS_TypeCode** (p. 2071) in IDL notation.

MT Safety:

SAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>indent</i>	<< <i>in</i> >> (p. 1013) Indent.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220)

5.11.7.46 DDS_TypeCode_print()

```
void DDS_TypeCode_print (
    const DDS_TypeCode * self,
    const DDS_TypeCodePrintFormatProperty * format,
    DDS_ExceptionCode_t * ex )
```

Prints a **DDS_TypeCode** (p. 2071).

MT Safety:

SAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>format</i>	<< <i>in</i> >> (p. 1013) Format.

Parameters

<i>ex</i>	<<out>> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220)

5.11.7.47 DDS_TypeCode_to_string()

```
void DDS_TypeCode_to_string (
    const   DDS_TypeCode * self,
    char * str,
    DDS_UnsignedLong * str_size,
    DDS_ExceptionCode_t * ex )
```

Get a string representation of this **DDS_TypeCode** (p. 2071) object using the default values for **DDS_TypeCodePrintFormatProperty** (p. 2073).

This function obtains a string representation of a **DDS_TypeCode** (p. 2071) object, it creates this string using the default values of **DDS_TypeCodePrintFormatProperty** (p. 2073).

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. 2071) 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. 1220) 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

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>str</i>	<< out >> (p. 1013) The char buffer that will be used to store the string representation of the DDS_TypeCode (p. 2071). If NULL then this function will store the required length of the buffer into the value pointed to by the <i>str_size</i> parameter.
<i>str_size</i>	<< inout >> (p. 1013) 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.
<i>ex</i>	<< out >> (p. 1013) Parameter used for error indications.

See also

DDS_TypeCode_print_IDL (p. 283)

DDS_TypeCode_print (p. 284)

DDS_TypeCode_to_string_w_format (p. 286)

5.11.7.48 DDS_TypeCode_to_string_w_format()

```
void DDS_TypeCode_to_string_w_format (
    const   DDS_TypeCode * self,
    char * str,
    DDS_UnsignedLong * str_size,
    const   DDS_TypeCodePrintFormatProperty * format,
    DDS_ExceptionCode_t * ex )
```

Get a string representation of this **DDS_TypeCode** (p. 2071) object using the format described by **DDS_TypeCodePrintFormatProperty** (p. 2073).

This function behaves in the same way as **DDS_TypeCode_to_string** (p. 285) but allows the user to specify the format of the string using **DDS_TypeCodePrintFormatProperty** (p. 2073).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>str</i>	<< <i>out</i> >> (p. 1013) The char buffer which will hold the string representation of the DDS_TypeCode (p. 2071).
<i>str_size</i>	<< <i>inout</i> >> (p. 1013) 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.
<i>format</i>	<< <i>in</i> >> (p. 1013) The DDS_TypeCodePrintFormatProperty (p. 2073) used to define the format of the string representation of the DDS_TypeCode (p. 2071).
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter used for error indications.

See also

DDS_TypeCode_print_IDL (p. 283)

DDS_TypeCode_print (p. 284)

DDS_TypeCode_to_string (p. 285)

5.11.7.49 DDS_TypeCode_get_cdr_serialized_sample_max_size()

```
DDS_UnsignedLong DDS_TypeCode_get_cdr_serialized_sample_max_size (
    const   DDS_TypeCode * self,
```

```
DDS_DataRepresentationId_t representation_id,
DDS_ExceptionCode_t * ex )
```

[DEPRECATED] Gets the maximum serialized size of samples of this type

DEPRECATED: use **DDS_TypeCode_cdr_serialized_sample_max_size** (p. 287) 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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>representation_id</i>	The serialized data representation for which we calculate the maximum size.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications.

Returns

The maximum size

5.11.7.50 DDS_TypeCode_cdr_serialized_sample_max_size()

```
DDS_UnsignedLong DDS_TypeCode_cdr_serialized_sample_max_size (
    const DDS_TypeCode * self,
    DDS_DataRepresentationId_t representation_id,
    DDS_ExceptionCode_t * ex )
```

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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>representation_id</i>	The serialized data representation for which we calculate the maximum size.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications.

Returns

The maximum size

5.11.7.51 DDS_TypeCode_cdr_serialized_sample_min_size()

```
DDS_UnsignedLong DDS_TypeCode_cdr_serialized_sample_min_size (
    const DDS_TypeCode * self,
    DDS_DataRepresentationId_t representation_id,
    DDS_ExceptionCode_t * ex )
```

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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>representation_id</i>	The serialized data representation for which we calculate the minimum size.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications.

Returns

The minimum size

5.11.7.52 DDS_TypeCode_cdr_serialized_sample_key_max_size()

```
DDS_UnsignedLong DDS_TypeCode_cdr_serialized_sample_key_max_size (
    const DDS_TypeCode * self,
    DDS_DataRepresentationId_t representation_id,
    DDS_ExceptionCode_t * ex )
```

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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>representation_id</i>	The serialized data representation for which we calculate the maximum key size.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications.

Returns

The maximum key size

5.11.7.53 DDS_TypeCodeFactory_get_instance()

```
DDS_TypeCodeFactory * DDS_TypeCodeFactory_get_instance (
    void )
```

Gets the singleton instance of this class.

Returns

The **DDS_TypeCodeFactory** (p. 2071) instance if no errors. Otherwise, NULL.

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this function while another thread may be simultaneously calling **DDS_DomainParticipantFactory_get_instance** (p. 38), **DDS_DomainParticipantFactory_finalize_instance** (p. 38), **DDS_TypeCodeFactory_get_instance** (p. 289), **DDS_TypeCodeFactory_finalize_instance** (p. 289), **NDDS_Utility_enable_network_capture**, or **NDDS_Utility_disable_network_capture**.

Examples

HelloWorldPlugin.c.

5.11.7.54 DDS_TypeCodeFactory_finalize_instance()

```
DDS_ReturnCode_t DDS_TypeCodeFactory_finalize_instance (
    void )
```

If there are no **DDS_DomainParticipantFactory** instances left in the system, it will finalize the **DDS_TypeCodeFactory** (p. 2071). If there is at least one **DDS_DomainParticipantFactory** instance left in the system, this is a noop.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if finalized.

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this function while another thread may be simultaneously calling **DDS_DomainParticipantFactory_get_instance** (p. 38), **DDS_DomainParticipantFactory_finalize_instance** (p. 38), **DDS_TypeCodeFactory_get_instance** (p. 289), **DDS_TypeCodeFactory_finalize_instance** (p. 289), **NDDS_Utility_enable_network_capture**, or **NDDS_Utility_disable_network_capture**.

5.11.7.55 DDS_TypeCodeFactory_clone_tc()

```
DDS_TypeCode * DDS_TypeCodeFactory_clone_tc (
    DDS_TypeCodeFactory * self,
    const DDS_TypeCode * tc,
    DDS_ExceptionCode_t * ex )
```

Creates and returns a copy of the input **DDS_TypeCode** (p. 2071).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>tc</i>	<< <i>in</i> >> (p. 1013) Type code that will be copied. Cannot be NULL.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220)

Returns

A clone of tc.

5.11.7.56 DDS_TypeCodeFactory_delete_tc()

```
void DDS_TypeCodeFactory_delete_tc (
    DDS_TypeCodeFactory * self,
    DDS_TypeCode * tc,
    DDS_ExceptionCode_t * ex )
```

Deletes the input **DDS_TypeCode** (p. 2071).

All the type codes created through the **DDS_TypeCodeFactory** (p. 2071) must be deleted using this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>tc</i>	<< <i>inout</i> >> (p. 1013) Type code that will be deleted. Cannot be NULL.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220)

Destroys a passed typecode.

5.11.7.57 DDS_TypeCodeFactory_get_primitive_tc()

```
const DDS_TypeCode * DDS_TypeCodeFactory_get_primitive_tc (
    DDS_TypeCodeFactory * self,
    DDS_TCKind tc_kind )
```

Get the **DDS_TypeCode** (p. 2071) for a primitive type (integers, floating point values, etc.) identified by the given **DDS_↵_TCKind** (p. 243).

This method is equivalent to, and replaces, the `DDS_g_tc_*` constants.

See also

DDS_g_tc_long (p. 301)
DDS_g_tc_ulong (p. 301)
DDS_g_tc_short (p. 300)
DDS_g_tc_ushort (p. 301)
DDS_g_tc_float (p. 302)
DDS_g_tc_double (p. 302)
DDS_g_tc_longdouble (p. 305)
DDS_g_tc_octet (p. 303)
DDS_g_tc_boolean (p. 302)
DDS_g_tc_char (p. 303)
DDS_g_tc_wchar (p. 305)

5.11.7.58 DDS_TypeCodeFactory_create_struct_tc()

```
DDS_TypeCode * DDS_TypeCodeFactory_create_struct_tc (
    DDS_TypeCodeFactory * self,
    const char * name,
    const struct DDS_StructMemberSeq * members,
    DDS_ExceptionCode_t * ex )
```

Constructs a **DDS_TK_STRUCTURE** (p. 243) **DDS_TypeCode** (p. 2071).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>name</i>	<< <i>in</i> >> (p. 1013) Name of the struct type. Cannot be NULL.
<i>members</i>	<< <i>in</i> >> (p. 1013) Initial members of the structure. This list may be empty (that is, FooSeq_get_length (p. 1515) 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

<i>ex</i>	<<out>> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220)

Returns

A newly-created **DDS_TypeCode** (p. 2071) object describing a struct.

5.11.7.59 DDS_TypeCodeFactory_create_struct_tc_ex()

```
DDS_TypeCode * DDS_TypeCodeFactory_create_struct_tc_ex (
    DDS_TypeCodeFactory * self,
    const char * name,
    DDS_ExtensibilityKind extensibility_kind,
    const struct DDS_StructMemberSeq * members,
    DDS_ExceptionCode_t * ex )
```

Constructs a **DDS_TK_STRUCT** (p. 243) **DDS_TypeCode** (p. 2071).

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>name</i>	<< in >> (p. 1013) Name of the struct type. Cannot be NULL.
<i>extensibility_kind</i>	<< in >> (p. 1013) Type extensibility.
<i>members</i>	<< in >> (p. 1013) Initial members of the structure. This list may be empty (that is, FooSeq_get_length (p. 1515) 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.)
<i>ex</i>	<<out>> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220)

Returns

A newly-created **DDS_TypeCode** (p. 2071) object describing a struct.

5.11.7.60 DDS_TypeCodeFactory_create_value_tc()

```
DDS_TypeCode * DDS_TypeCodeFactory_create_value_tc (
    DDS_TypeCodeFactory * self,
    const char * name,
    DDS_ValueModifier type_modifier,
    const DDS_TypeCode * concrete_base,
    const struct DDS_ValueMemberSeq * members,
    DDS_ExceptionCode_t * ex )
```

Constructs a **DDS_TK_VALUE** (p. 243) **DDS_TypeCode** (p. 2071).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>name</i>	<< <i>in</i> >> (p. 1013) Name of the value type. Cannot be NULL.
<i>type_modifier</i>	<< <i>in</i> >> (p. 1013) One of the value type modifier constants: DDS_VM_NONE (p. 238), DDS_VM_CUSTOM (p. 238), DDS_VM_ABSTRACT (p. 239) or DDS_VM_TRUNCATABLE (p. 239).
<i>concrete_base</i>	<< <i>in</i> >> (p. 1013) DDS_TypeCode (p. 2071) object describing the concrete valuetype base. It may be NULL if the valuetype does not have a concrete base.
<i>members</i>	<< <i>in</i> >> (p. 1013) 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.)
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220)

Returns

A newly-created **DDS_TypeCode** (p. 2071) object describing a value.

5.11.7.61 DDS_TypeCodeFactory_create_value_tc_ex()

```
DDS_TypeCode * DDS_TypeCodeFactory_create_value_tc_ex (
    DDS_TypeCodeFactory * self,
```

```

const char * name,
    DDS_ExtensibilityKind extensibility_kind,
    DDS_ValueModifier type_modifier,
const    DDS_TypeCode * concrete_base,
const struct    DDS_ValueMemberSeq * members,
    DDS_ExceptionCode_t * ex )

```

Constructs a **DDS_TK_VALUE** (p. 243) **DDS_TypeCode** (p. 2071).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>name</i>	<< <i>in</i> >> (p. 1013) Name of the value type. Cannot be NULL.
<i>extensibility_kind</i>	<< <i>in</i> >> (p. 1013) Type extensibility.
<i>type_modifier</i>	<< <i>in</i> >> (p. 1013) One of the value type modifier constants: DDS_VM_NONE (p. 238), DDS_VM_CUSTOM (p. 238), DDS_VM_ABSTRACT (p. 239) or DDS_VM_TRUNCATABLE (p. 239).
<i>concrete_base</i>	<< <i>in</i> >> (p. 1013) DDS_TypeCode (p. 2071) object describing the concrete valuetype base. It may be NULL if the valuetype does not have a concrete base.
<i>members</i>	<< <i>in</i> >> (p. 1013) 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.)
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220)

Returns

A newly-created **DDS_TypeCode** (p. 2071) object describing a value.

5.11.7.62 DDS_TypeCodeFactory_create_union_tc()

```

DDS_TypeCode * DDS_TypeCodeFactory_create_union_tc (
    DDS_TypeCodeFactory * self,
const char * name,
const    DDS_TypeCode * discriminator_type,
    DDS_Long default_index,
const struct    DDS_UnionMemberSeq * members,
    DDS_ExceptionCode_t * ex )

```

Constructs a **DDS_TK_UNION** (p. 243) **DDS_TypeCode** (p. 2071).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>name</i>	<< <i>in</i> >> (p. 1013) Name of the union type. Cannot be NULL.
<i>discriminator_type</i>	<< <i>in</i> >> (p. 1013) Discriminator Type Code. Cannot be NULL.
<i>default_index</i>	<< <i>in</i> >> (p. 1013) Index of the default member, or -1 if there is no default member.
<i>members</i>	<< <i>in</i> >> (p. 1013) 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.)
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220)

Returns

A newly-created **DDS_TypeCode** (p. 2071) object describing a union.

5.11.7.63 DDS_TypeCodeFactory_create_union_tc_ex()

```

DDS_TypeCode * DDS_TypeCodeFactory_create_union_tc_ex (
    DDS_TypeCodeFactory * self,
    const char * name,
    DDS_ExtensibilityKind extensibility_kind,
    const DDS_TypeCode * discriminator_type,
    DDS_Long default_index,
    const struct DDS_UnionMemberSeq * members,
    DDS_ExceptionCode_t * ex )

```

Constructs a **DDS_TK_UNION** (p. 243) **DDS_TypeCode** (p. 2071).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>name</i>	<< <i>in</i> >> (p. 1013) Name of the union type. Cannot be NULL.
<i>extensibility_kind</i>	<< <i>in</i> >> (p. 1013) Type extensibility.
<i>discriminator_type</i>	<< <i>in</i> >> (p. 1013) Discriminator Type Code. Cannot be NULL.
<i>default_index</i>	<< <i>in</i> >> (p. 1013) Index of the default member, or -1 if there is no default member.
<i>members</i>	<< <i>in</i> >> (p. 1013) 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.)

Parameters

<i>ex</i>	<<out>> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220)

Returns

A newly-created **DDS_TypeCode** (p. 2071) object describing a union.

5.11.7.64 DDS_TypeCodeFactory_create_enum_tc()

```
DDS_TypeCode * DDS_TypeCodeFactory_create_enum_tc (
    DDS_TypeCodeFactory * self,
    const char * name,
    const struct DDS_EnumMemberSeq * members,
    DDS_ExceptionCode_t * ex )
```

Constructs a **DDS_TK_ENUM** (p. 243) **DDS_TypeCode** (p. 2071).

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>name</i>	<< in >> (p. 1013) Name of the enum type. Cannot be NULL.
<i>members</i>	<< in >> (p. 1013) 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. 278).
<i>ex</i>	<<out>> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE (p. 1220)

Returns

A newly-created **DDS_TypeCode** (p. 2071) object describing an enumeration.

5.11.7.65 DDS_TypeCodeFactory_create_enum_tc_ex()

```

DDS_TypeCode * DDS_TypeCodeFactory_create_enum_tc_ex (
    DDS_TypeCodeFactory * self,
    const char * name,
    DDS_ExtensibilityKind extensibility_kind,
    const struct DDS_EnumMemberSeq * members,
    DDS_ExceptionCode_t * ex )

```

Constructs a **DDS_TK_ENUM** (p. 243) **DDS_TypeCode** (p. 2071).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>name</i>	<< <i>in</i> >> (p. 1013) Name of the enum type. Cannot be NULL.
<i>extensibility_kind</i>	<< <i>in</i> >> (p. 1013) Type extensibility.
<i>members</i>	<< <i>in</i> >> (p. 1013) 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. 278).
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE (p. 1220)

Returns

A newly-created **DDS_TypeCode** (p. 2071) object describing an enumeration.

5.11.7.66 DDS_TypeCodeFactory_create_alias_tc()

```

DDS_TypeCode * DDS_TypeCodeFactory_create_alias_tc (
    DDS_TypeCodeFactory * self,
    const char * name,
    const DDS_TypeCode * original_type,
    DDS_Boolean is_pointer,
    DDS_ExceptionCode_t * ex )

```

Constructs a **DDS_TK_ALIAS** (p. 243) (typedef) **DDS_TypeCode** (p. 2071).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>name</i>	<< <i>in</i> >> (p. 1013) Name of the alias. Cannot be NULL.

Parameters

<i>original_type</i>	<< <i>in</i> >> (p. 1013) Aliased type code. Cannot be NULL.
<i>is_pointer</i>	<< <i>in</i> >> (p. 1013) Indicates if the alias is a pointer to the aliased type code.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220)

Returns

A newly-created **DDS_TypeCode** (p. 2071) object describing an alias.

5.11.7.67 DDS_TypeCodeFactory_create_string_tc()

```
DDS_TypeCode * DDS_TypeCodeFactory_create_string_tc (
    DDS_TypeCodeFactory * self,
    DDS_UnsignedLong bound,
    DDS_ExceptionCode_t * ex )
```

Constructs a **DDS_TK_STRING** (p. 243) **DDS_TypeCode** (p. 2071).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>bound</i>	<< <i>in</i> >> (p. 1013) Maximum length of the string.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE (p. 1220)

Returns

A newly-created **DDS_TypeCode** (p. 2071) object describing a string.

5.11.7.68 DDS_TypeCodeFactory_create_wstring_tc()

```
DDS_TypeCode * DDS_TypeCodeFactory_create_wstring_tc (
    DDS_TypeCodeFactory * self,
    DDS_UnsignedLong bound,
    DDS_ExceptionCode_t * ex )
```

Constructs a **DDS_TK_WSTRING** (p. 243) **DDS_TypeCode** (p. 2071).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>bound</i>	<< <i>in</i> >> (p. 1013) Maximum length of the wide string.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE (p. 1220)

Returns

A newly-created **DDS_TypeCode** (p. 2071) object describing a wide string.

5.11.7.69 DDS_TypeCodeFactory_create_sequence_tc()

```
DDS_TypeCode * DDS_TypeCodeFactory_create_sequence_tc (
    DDS_TypeCodeFactory * self,
    DDS_UnsignedLong bound,
    const DDS_TypeCode * element_type,
    DDS_ExceptionCode_t * ex )
```

Constructs a **DDS_TK_SEQUENCE** (p. 243) **DDS_TypeCode** (p. 2071).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>bound</i>	<< <i>in</i> >> (p. 1013) The bound for the sequence (> 0).
<i>element_type</i>	<< <i>in</i> >> (p. 1013) DDS_TypeCode (p. 2071) object describing the sequence elements.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220)
Generated by Doxygen	• DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220)

Returns

A newly-created **DDS_TypeCode** (p. 2071) object describing a sequence.

5.11.7.70 DDS_TypeCodeFactory_create_array_tc()

```
DDS_TypeCode * DDS_TypeCodeFactory_create_array_tc (
    DDS_TypeCodeFactory * self,
    const struct DDS_UnsignedLongSeq * dimensions,
    const DDS_TypeCode * element_type,
    DDS_StatusCode_t * ex )
```

Constructs a **DDS_TK_ARRAY** (p. 243) **DDS_TypeCode** (p. 2071).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>dimensions</i>	<< <i>in</i> >> (p. 1013) Dimensions of the array. Each dimension has to be greater than 0.
<i>element_type</i>	<< <i>in</i> >> (p. 1013) DDS_TypeCode (p. 2071) describing the array elements. Cannot be NULL.
<i>ex</i>	<< <i>out</i> >> (p. 1013) Parameter for error indications. The values that can take are: • DDS_NO_EXCEPTION_CODE (p. 1219) • DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE (p. 1220) • DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE (p. 1220)

Returns

A newly-created **DDS_TypeCode** (p. 2071) object describing a sequence.

5.11.8 Variable Documentation

5.11.8.1 DDS_g_tc_null

```
DDS_TypeCode DDS_g_tc_null
```

Basic NULL type.

For new code, **DDS_TypeCodeFactory_get_primitive_tc** (p. 291) is preferred to using this global variable.

See also

DDS_TypeCodeFactory_get_primitive_tc (p. 291)

5.11.8.2 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. 291) is preferred to using this global variable.

See also

DDS_TypeCodeFactory_get_primitive_tc (p. 291)

DDS_Short (p. 1202)

5.11.8.3 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. 291) is preferred to using this global variable.

See also

DDS_TypeCodeFactory_get_primitive_tc (p. 291)

DDS_Long (p. 1203)

5.11.8.4 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. 291) is preferred to using this global variable.

See also

DDS_TypeCodeFactory_get_primitive_tc (p. 291)

DDS_UnsignedShort (p. 1202)

5.11.8.5 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. 291) is preferred to using this global variable.

See also

DDS_TypeCodeFactory_get_primitive_tc (p. 291)

DDS_UnsignedLong (p. 1203)

5.11.8.6 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. 291) is preferred to using this global variable.

See also

DDS_TypeCodeFactory_get_primitive_tc (p. 291)

DDS_Float (p. 1203)

5.11.8.7 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. 291) is preferred to using this global variable.

See also

DDS_TypeCodeFactory_get_primitive_tc (p. 291)

DDS_Double (p. 1203)

5.11.8.8 DDS_g_tc_boolean

`DDS_TypeCode DDS_g_tc_boolean`

Basic Boolean type.

For new code, `DDS_TypeCodeFactory_get_primitive_tc` (p. 291) is preferred to using this global variable.

See also

`DDS_TypeCodeFactory_get_primitive_tc` (p. 291)

`DDS_Boolean` (p. 1204)

5.11.8.9 DDS_g_tc_char

`DDS_TypeCode DDS_g_tc_char`

Basic single-byte character type.

For new code, `DDS_TypeCodeFactory_get_primitive_tc` (p. 291) is preferred to using this global variable.

See also

`DDS_TypeCodeFactory_get_primitive_tc` (p. 291)

`DDS_Char` (p. 1201)

5.11.8.10 DDS_g_tc_octet

`DDS_TypeCode DDS_g_tc_octet`

Basic octet/byte type.

For new code, `DDS_TypeCodeFactory_get_primitive_tc` (p. 291) is preferred to using this global variable.

See also

`DDS_TypeCodeFactory_get_primitive_tc` (p. 291)

`DDS_Octet` (p. 1202)

5.11.8.11 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. 291) is preferred to using this global variable.

See also

DDS_TypeCodeFactory_get_primitive_tc (p. 291)

DDS_Int8 (p. 1202)

5.11.8.12 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. 291) is preferred to using this global variable.

See also

DDS_TypeCodeFactory_get_primitive_tc (p. 291)

DDS_UInt8 (p. 1202)

5.11.8.13 DDS_g_tc_longlong

DDS_TypeCode DDS_g_tc_longlong

Basic 64-bit integer type.

For new code, **DDS_TypeCodeFactory_get_primitive_tc** (p. 291) is preferred to using this global variable.

See also

DDS_TypeCodeFactory_get_primitive_tc (p. 291)

DDS_LongLong (p. 1203)

5.11.8.14 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. 291) is preferred to using this global variable.

See also

DDS_TypeCodeFactory_get_primitive_tc (p. 291)

DDS_UnsignedLongLong (p. 1203)

5.11.8.15 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. 291) is preferred to using this global variable.

See also

DDS_TypeCodeFactory_get_primitive_tc (p. 291)

DDS_LongDouble (p. 1204)

5.11.8.16 DDS_g_tc_wchar

DDS_TypeCode DDS_g_tc_wchar

Basic four-byte character type.

For new code, **DDS_TypeCodeFactory_get_primitive_tc** (p. 291) is preferred to using this global variable.

See also

DDS_TypeCodeFactory_get_primitive_tc (p. 291)

DDS_Wchar (p. 1202)

5.12 Built-in Types

RTI Connex 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.

5.12.1 Detailed Description

RTI Connex 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. 1812): 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. 1859): A payload consisting of an opaque variable-length array of bytes. This type has no key.
- **DDS_KeyedOctets** (p. 1811): 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. 1812) types are appropriate for simple text-based applications. The **DDS_Octets** (p. 1859) and **DDS_KeyedOctets** (p. 1811) types are appropriate for applications that perform their own custom data serialization, such as legacy applications still in the process of migrating to RTI Connex. 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 **DDS_ContentFilteredTopic** (p. 177)).

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. 313) API

5.12.2 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. 1901) 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 `DomainParticipant` unless they are overwritten in the `DataWriters` and `DataReaders`.

Table 5.293 *Builtin Types Allocation Properties*

Property	Description
dds.builtin_type.string.alloc_size	Maximum size of the strings published by the DDS_StringDataWriter (p. 1114) or received by the DDS_StringDataReader (p. 1114) (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 DDS_KeyedStringDataWriter (p. 1129) or DDS_KeyedStringDataReader (p. 1129) (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 DDS_KeyedStringDataWriter (p. 1129) or received by the DDS_KeyedStringDataReader (p. 1129) (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 by the DDS_OctetsDataWriter (p. 1154) or received by the DDS_OctetsDataReader (p. 1154). 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 DDS_KeyedOctetsDataWriter (p. 1175) or received by the DDS_KeyedOctetsDataReader (p. 1175) (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 DDS_KeyedOctetsDataWriter (p. 1175) or received by a DDS_KeyedOctetsDataReader (p. 1175). 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.

5.12.3 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 5.294 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*.

5.13 Built-in Topic's Trust Types

Types used as part of **DDS_ParticipantBuiltinTopicData** (p. 1870), **DDS_PublicationBuiltinTopicData** (p. 1905), **DDS_SubscriptionBuiltinTopicData** (p. 2012) to describe Trust Plugins configuration.

Data Structures

- 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_ParticipantTrustKeyEstablishmentAlgorithmInfo** ↔
Trust Plugins key establishment algorithm information associated with the discovered DomainParticipant.

- `typedef struct DDS_ParticipantTrustInterceptorAlgorithmInfo DDS_ParticipantTrustInterceptorAlgorithmInfo`
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.

5.13.1 Detailed Description

Types used as part of **DDS_ParticipantBuiltinTopicData** (p. 1870), **DDS_PublicationBuiltinTopicData** (p. 1905), **DDS_SubscriptionBuiltinTopicData** (p. 2012) 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*.

5.13.2 Typedef Documentation

5.13.2.1 DDS_ParticipantTrustAttributesMask

```
typedef DDS_UnsignedLong DDS_ParticipantTrustAttributesMask
```

5.13.2.2 DDS_PluginParticipantTrustAttributesMask

```
typedef DDS_UnsignedLong DDS_PluginParticipantTrustAttributesMask
```

5.13.2.3 DDS_ParticipantTrustProtectionInfo

```
typedef struct DDS_ParticipantTrustProtectionInfo DDS_ParticipantTrustProtectionInfo
```

Trust Plugins Protection information associated with the discovered DomainParticipant.

5.13.2.4 DDS_EndpointTrustAttributesMask

```
typedef DDS_UnsignedLong DDS_EndpointTrustAttributesMask
```

5.13.2.5 DDS_PluginEndpointTrustAttributesMask

```
typedef DDS_UnsignedLong DDS_PluginEndpointTrustAttributesMask
```

5.13.2.6 DDS_VendorEndpointTrustAttributesMask

```
typedef DDS_UnsignedLong DDS_VendorEndpointTrustAttributesMask
```

5.13.2.7 DDS_EndpointTrustProtectionInfo

```
typedef struct DDS_EndpointTrustProtectionInfo DDS_EndpointTrustProtectionInfo
```

Trust Plugins Protection information associated with the discovered endpoint.

5.13.2.8 DDS_TrustAlgorithmBit

```
typedef DDS_UnsignedLong DDS_TrustAlgorithmBit
```

5.13.2.9 DDS_TrustAlgorithmSet

```
typedef DDS_UnsignedLong DDS_TrustAlgorithmSet
```

5.13.2.10 DDS_TrustAlgorithmRequirements

```
typedef struct DDS_TrustAlgorithmRequirements DDS_TrustAlgorithmRequirements
```

Type to describe Trust Plugins algorithm requirements for an entity.

5.13.2.11 DDS_ParticipantTrustSignatureAlgorithmInfo

```
typedef struct DDS_ParticipantTrustSignatureAlgorithmInfo DDS_ParticipantTrustSignatureAlgorithmInfo
```

Trust Plugins signature algorithm information associated with the discovered DomainParticipant.

5.13.2.12 DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo

```
typedef struct DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo
```

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

5.13.2.13 DDS_ParticipantTrustInterceptorAlgorithmInfo

```
typedef struct DDS_ParticipantTrustInterceptorAlgorithmInfo DDS_ParticipantTrustInterceptorAlgorithmInfo
```

Trust Plugins interception algorithm information associated with the discovered DomainParticipant.

5.13.2.14 DDS_ParticipantTrustAlgorithmInfo

```
typedef struct DDS_ParticipantTrustAlgorithmInfo DDS_ParticipantTrustAlgorithmInfo
```

Trust Plugins algorithm information associated with the discovered DomainParticipant.

5.13.2.15 DDS_EndpointTrustInterceptorAlgorithmInfo

```
typedef struct DDS_EndpointTrustInterceptorAlgorithmInfo DDS_EndpointTrustInterceptorAlgorithmInfo
```

Trust Plugins interception algorithm information associated with the discovered endpoint.

5.13.2.16 DDS_EndpointTrustAlgorithmInfo

```
typedef struct DDS_EndpointTrustAlgorithmInfo DDS_EndpointTrustAlgorithmInfo
```

Trust Plugins algorithm information associated with the discovered endpoint.

5.14 Dynamic Data

<<*extension*>> (p. 1012) The Dynamic Data API provides a way to interact with arbitrarily complex data types at runtime without the need for code generation.

Data Structures

- struct **DDS_DynamicDataProperty_t**
*A collection of attributes used to configure **DDS_DynamicData** (p. 1768) 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. 1768) 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. 1768) elements.*
- struct **DDS_DynamicDataTypeProperty_t**
*A collection of attributes used to configure **DDS_DynamicDataTypeSupport** (p. 326) objects.*
- struct **DDS_DynamicDataJsonParserProperties_t**
*A collection of attributes used to configure **DDS_DynamicData** (p. 1768) objects.*

Macros

- #define **DDS_DynamicDataProperty_t_INITIALIZER**
*Static initializer for **DDS_DynamicDataProperty_t** (p. 1779) objects.*
- #define **DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED**
A sentinel value that indicates that no member ID is needed in order to perform some operation.
- #define **DDS_DynamicDataTypeProperty_t_INITIALIZER**
*Static initializer for **DDS_DynamicDataTypeProperty_t** (p. 1781) objects.*

Typedefs

- typedef **DDS_Long DDS_DynamicDataMemberId**
An integer that uniquely identifies some member of a data type within that type.
- typedef struct **DDS_DynamicDataWriter DDS_DynamicDataWriter**
*Writes (publishes) objects of type **DDS_DynamicData** (p. 1768).*
- typedef struct **DDS_DynamicDataReader DDS_DynamicDataReader**
*Reads (subscribes to) objects of type **DDS_DynamicData** (p. 1768).*
- typedef struct **DDS_DynamicDataTypeSupport DDS_DynamicDataTypeSupport**
*A factory for registering a dynamically defined type and creating **DDS_DynamicData** (p. 1768) objects.*

Functions

- **DDS_Boolean DDS_DynamicData_initialize** (**DDS_DynamicData** *self, const **DDS_TypeCode** *type, const struct **DDS_DynamicDataProperty_t** *property)
*Initialize a **DDS_DynamicData** (p. 1768) object to a valid empty state.*
- **DDS_DynamicData * DDS_DynamicData_new** (const **DDS_TypeCode** *type, const struct **DDS_DynamicDataProperty_t** *property)
*The constructor for new **DDS_DynamicData** (p. 1768) objects.*
- void **DDS_DynamicData_finalize** (**DDS_DynamicData** *self)
Clean up the internal state of this object to get it ready for deallocation, but don't deallocate the pointer.
- void **DDS_DynamicData_delete** (**DDS_DynamicData** *self)
*Finalize and deallocate this **DDS_DynamicData** (p. 1768) sample.*
- **DDS_ReturnCode_t DDS_DynamicData_copy** (**DDS_DynamicData** *self, const **DDS_DynamicData** *src)
Deeply copy from the given object to this object.
- **DDS_Boolean DDS_DynamicData_equal** (const **DDS_DynamicData** *self, const **DDS_DynamicData** *other)
*Indicate whether the contents of another **DDS_DynamicData** (p. 1768) sample are the same as those of this one.*
- **DDS_ReturnCode_t DDS_DynamicData_clear_all_members** (**DDS_DynamicData** *self)
Clear the contents of all data members of this object.
- **DDS_ReturnCode_t DDS_DynamicData_clear_optional_member** (**DDS_DynamicData** *self, const char *member_name, **DDS_DynamicDataMemberId** member_id)
Clear the contents of a single optional data member of this object.
- **DDS_ReturnCode_t DDS_DynamicData_clear_member** (**DDS_DynamicData** *self, const char *member_name, **DDS_DynamicDataMemberId** member_id)
Clear the contents of a single data member of this object.
- **DDS_ReturnCode_t DDS_DynamicData_set_cdr_buffer** (**DDS_DynamicData** *self, const char *buffer, **DDS_UnsignedLong** length)
Associate a CDR buffer with a DynamicData object.
- const char * **DDS_DynamicData_get_cdr_buffer** (const **DDS_DynamicData** *self, **DDS_UnsignedLong** *length)
Get the CDR buffer associated with a DynamicData object.
- **DDS_Boolean DDS_DynamicData_is_cdr** (const **DDS_DynamicData** *self)
Determine if a DynamicData object is using CDR as its data representation.
- **DDS_ReturnCode_t DDS_DynamicData_set_discriminator** (**DDS_DynamicData** *self, **DDS_Long** discriminator)
Set the value of the union discriminator (UnionType only)

- **DDS_ReturnCode_t DDS_DynamicData_get_discriminator** (const **DDS_DynamicData** *self, **DDS_Long** *discriminator)
Get the current value of the union discriminator.
- **DDS_ReturnCode_t DDS_DynamicData_print** (const **DDS_DynamicData** *self, FILE *fp, int indent)
Output a textual representation of this object and its contents to the given file.
- **DDS_ReturnCode_t DDS_DynamicData_from_cdr_buffer** (**DDS_DynamicData** *self, const char *buffer, unsigned int length)
Deserializes a DynamicData object from a buffer of octets.
- **DDS_ReturnCode_t DDS_DynamicData_to_cdr_buffer_ex** (**DDS_DynamicData** *self, char *buffer, unsigned int *length, **DDS_DataRepresentationId_t** representation)
Serializes a DynamicData object into a buffer of octets.
- **DDS_ReturnCode_t DDS_DynamicData_to_cdr_buffer** (**DDS_DynamicData** *self, char *buffer, unsigned int *length)
Serializes a DynamicData object into a CDR buffer of octets.
- **DDS_ReturnCode_t DDS_DynamicData_to_string** (**DDS_DynamicData** *self, char *str, **DDS_UnsignedLong** *str_size, const struct **DDS_PrintFormatProperty** *property)
Get a string representation of a DynamicData object.
- **DDS_ReturnCode_t DDS_DynamicData_from_string** (**DDS_DynamicData** *self, const char *str, **DDS_PrintFormatKind** format_kind)
Populates a DynamicData object from a JSON string representation.
- **void DDS_DynamicData_get_info** (const **DDS_DynamicData** *self, struct **DDS_DynamicDataInfo** *info_out)
*Fill in the given descriptor with information about this **DDS_DynamicData** (p. 1768).*
- **DDS_ReturnCode_t DDS_DynamicData_bind_type** (**DDS_DynamicData** *self, const **DDS_TypeCode** *type)
*If this **DDS_DynamicData** (p. 1768) object is not yet associated with a data type, set that type now to the given **DDS_TypeCode** (p. 2071).*
- **DDS_ReturnCode_t DDS_DynamicData_unbind_type** (**DDS_DynamicData** *self)
*Dissociate this **DDS_DynamicData** (p. 1768) object from any particular data type.*
- **DDS_ReturnCode_t DDS_DynamicData_bind_complex_member** (**DDS_DynamicData** *self, **DDS_DynamicData** *value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id)
*Use another **DDS_DynamicData** (p. 1768) object to provide access to a complex field of this **DDS_DynamicData** (p. 1768) object.*
- **DDS_ReturnCode_t DDS_DynamicData_unbind_complex_member** (**DDS_DynamicData** *self, **DDS_DynamicData** *value)
*Tear down the association created by a **DDS_DynamicData_bind_complex_member** (p. 342) operation, committing any changes to the outer object since then.*
- **const DDS_TypeCode * DDS_DynamicData_get_type** (const **DDS_DynamicData** *self)
*Get the data type, of which this **DDS_DynamicData** (p. 1768) represents an instance.*
- **DDS_TCKind DDS_DynamicData_get_type_kind** (const **DDS_DynamicData** *self)
Get the kind of this object's data type.
- **DDS_UnsignedLong DDS_DynamicData_get_member_count** (const **DDS_DynamicData** *self)
Get the number of members in the type.
- **DDS_Boolean DDS_DynamicData_member_exists** (const **DDS_DynamicData** *self, const char *member_name, **DDS_DynamicDataMemberId** member_id)
Indicates whether a member exists in this sample.
- **DDS_Boolean DDS_DynamicData_member_exists_in_type** (const **DDS_DynamicData** *self, const char *member_name, **DDS_DynamicDataMemberId** member_id)
Indicates whether a member of a particular name/ID exists in this data sample's type.

- **DDS_ReturnCode_t DDS_DynamicData_get_member_info** (const **DDS_DynamicData** *self, struct **DDS_DynamicDataMemberInfo** *info, const char *member_name, **DDS_DynamicDataMemberId** member_id)
*Fill in the given descriptor with information about the identified member of this **DDS_DynamicData** (p. 1768) sample.*
- **DDS_ReturnCode_t DDS_DynamicData_get_member_info_by_index** (const **DDS_DynamicData** *self, struct **DDS_DynamicDataMemberInfo** *info, **DDS_UnsignedLong** index)
*Fill in the given descriptor with information about the identified member of this **DDS_DynamicData** (p. 1768) sample.*
- **DDS_ReturnCode_t DDS_DynamicData_get_member_type** (const **DDS_DynamicData** *self, const **DDS_TypeCode** **type_out, const char *member_name, **DDS_DynamicDataMemberId** member_id)
Get the type of the given member of this sample.
- **DDS_ReturnCode_t DDS_DynamicData_is_member_key** (const **DDS_DynamicData** *self, **DDS_Boolean** *is_key_out, const char *member_name, **DDS_DynamicDataMemberId** member_id)
Indicates whether a given member forms part of the key of this sample's data type.
- **DDS_ReturnCode_t DDS_DynamicData_get_long** (const **DDS_DynamicData** *self, **DDS_Long** *value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id)
*Get the value of the given field, which is of type **DDS_Long** (p. 1203) or another type implicitly convertible to it (**DDS_Octet** (p. 1202), **DDS_Char** (p. 1201), **DDS_Short** (p. 1202), **DDS_UnsignedShort** (p. 1202), or **DDS_Enum** (p. 1204)).*
- **DDS_ReturnCode_t DDS_DynamicData_get_short** (const **DDS_DynamicData** *self, **DDS_Short** *value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id)
*Get the value of the given field, which is of type **DDS_Short** (p. 1202) or another type implicitly convertible to it (**DDS_Octet** (p. 1202) or **DDS_Char** (p. 1201)).*
- **DDS_ReturnCode_t DDS_DynamicData_get_ulong** (const **DDS_DynamicData** *self, **DDS_UnsignedLong** *value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id)
*Get the value of the given field, which is of type **DDS_UnsignedLong** (p. 1203) or another type implicitly convertible to it (**DDS_Octet** (p. 1202), **DDS_Char** (p. 1201), **DDS_Short** (p. 1202), **DDS_UnsignedShort** (p. 1202), or **DDS_Enum** (p. 1204)).*
- **DDS_ReturnCode_t DDS_DynamicData_get_ushort** (const **DDS_DynamicData** *self, **DDS_UnsignedShort** *value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id)
*Get the value of the given field, which is of type **DDS_UnsignedShort** (p. 1202) or another type implicitly convertible to it (**DDS_Octet** (p. 1202) or **DDS_Char** (p. 1201)).*
- **DDS_ReturnCode_t DDS_DynamicData_get_float** (const **DDS_DynamicData** *self, **DDS_Float** *value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id)
*Get the value of the given field, which is of type **DDS_Float** (p. 1203).*
- **DDS_ReturnCode_t DDS_DynamicData_get_double** (const **DDS_DynamicData** *self, **DDS_Double** *value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id)
*Get the value of the given field, which is of type **DDS_Double** (p. 1203) or another type implicitly convertible to it (**DDS_Float** (p. 1203)).*
- **DDS_ReturnCode_t DDS_DynamicData_get_boolean** (const **DDS_DynamicData** *self, **DDS_Boolean** *value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id)
*Get the value of the given field, which is of type **DDS_Boolean** (p. 1204).*
- **DDS_ReturnCode_t DDS_DynamicData_get_char** (const **DDS_DynamicData** *self, **DDS_Char** *value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id)
*Get the value of the given field, which is of type **DDS_Char** (p. 1201).*
- **DDS_ReturnCode_t DDS_DynamicData_get_octet** (const **DDS_DynamicData** *self, **DDS_Octet** *value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id)
*Get the value of the given field, which is of type **DDS_Octet** (p. 1202).*
- **DDS_ReturnCode_t DDS_DynamicData_get_longlong** (const **DDS_DynamicData** *self, **DDS_LongLong** *value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id)
*Get the value of the given field, which is of type **DDS_LongLong** (p. 1203) or another type implicitly convertible to it (**DDS_Octet** (p. 1202), **DDS_Char** (p. 1201), **DDS_Short** (p. 1202), **DDS_UnsignedShort** (p. 1202), **DDS_Long** (p. 1203), **DDS_UnsignedLong** (p. 1203), or **DDS_Enum** (p. 1204)).*

- **DDS_ReturnCode_t DDS_DynamicData_get_ulonglong** (const **DDS_DynamicData** *self, **DDS_↵**
UnsignedLongLong *value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id)
*Get the value of the given field, which is of type **DDS_UnsignedLongLong** (p. 1203) or another type implicitly convertible to it (**DDS_Octet** (p. 1202), **DDS_Char** (p. 1201), **DDS_Short** (p. 1202), **DDS_UnsignedShort** (p. 1202), **DDS_Long** (p. 1203), **DDS_UnsignedLong** (p. 1203), or **DDS_Enum** (p. 1204)).*
- **DDS_ReturnCode_t DDS_DynamicData_get_longdouble** (const **DDS_DynamicData** *self, **DDS_Long↵**
Double *value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id)
*Get the value of the given field, which is of type **DDS_LongDouble** (p. 1204) or another type implicitly convertible to it (**DDS_Float** (p. 1203) or **DDS_Double** (p. 1203)).*
- **DDS_ReturnCode_t DDS_DynamicData_get_wchar** (const **DDS_DynamicData** *self, **DDS_Wchar** *value↵
_out, const char *member_name, **DDS_DynamicDataMemberId** member_id)
*Get the value of the given field, which is of type **DDS_Wchar** (p. 1202) or another type implicitly convertible to it (**DDS_↵**
Char (p. 1201)).*
- **DDS_ReturnCode_t DDS_DynamicData_get_int8** (const **DDS_DynamicData** *self, **DDS_Int8** *value_out,
const char *member_name, **DDS_DynamicDataMemberId** member_id)
*Get the value of the given field, which is of type **DDS_Int8** (p. 1202).*
- **DDS_ReturnCode_t DDS_DynamicData_get_uint8** (const **DDS_DynamicData** *self, **DDS_UInt8** *value_↵
out, const char *member_name, **DDS_DynamicDataMemberId** member_id)
*Get the value of the given field, which is of type **DDS_UInt8** (p. 1202).*
- **DDS_ReturnCode_t DDS_DynamicData_get_string** (const **DDS_DynamicData** *self, char **value, **DDS_↵**
UnsignedLong *size, const char *member_name, **DDS_DynamicDataMemberId** member_id)
Get the value of the given field, which is of type char.*
- **DDS_ReturnCode_t DDS_DynamicData_get_wstring** (const **DDS_DynamicData** *self, **DDS_Wchar**
value, **DDS_UnsignedLong *size, const char *member_name, **DDS_DynamicDataMemberId** member↵
_id)
*Get the value of the given field, which is of type **DDS_Wchar** (p. 1202)*.*
- **DDS_ReturnCode_t DDS_DynamicData_get_complex_member** (const **DDS_DynamicData** *self, **DDS_↵**
DynamicData *value_out, const char *member_name, **DDS_DynamicDataMemberId** member_id)
Get a copy of the value of the given field, which is of some composed type.
- **DDS_ReturnCode_t DDS_DynamicData_get_long_array** (const **DDS_DynamicData** *self, **DDS_Long**
*array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id)
*Get a copy of the given array member. The array may contain members of type **DDS_Long** (p. 1203) or **DDS_Enum** (p. 1204).*
- **DDS_ReturnCode_t DDS_DynamicData_get_short_array** (const **DDS_DynamicData** *self, **DDS_Short**
*array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id)
Get a copy of the given array member.
- **DDS_ReturnCode_t DDS_DynamicData_get_ulong_array** (const **DDS_DynamicData** *self, **DDS_↵**
UnsignedLong *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicData↵**
MemberId member_id)
*Get a copy of the given array member. The array may contain members of type **DDS_UnsignedLong** (p. 1203) or **DDS_↵**
Enum (p. 1204).*
- **DDS_ReturnCode_t DDS_DynamicData_get_ushort_array** (const **DDS_DynamicData** *self, **DDS_↵**
UnsignedShort *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicData↵**
MemberId member_id)
Get a copy of the given array member.
- **DDS_ReturnCode_t DDS_DynamicData_get_float_array** (const **DDS_DynamicData** *self, **DDS_Float**
*array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id)
Get a copy of the given array member.
- **DDS_ReturnCode_t DDS_DynamicData_get_double_array** (const **DDS_DynamicData** *self, **DDS_Double**
*array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given array member.

- **DDS_ReturnCode_t DDS_DynamicData_get_boolean_array** (const **DDS_DynamicData** *self, **DDS_Boolean** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given array member.

- **DDS_ReturnCode_t DDS_DynamicData_get_char_array** (const **DDS_DynamicData** *self, **DDS_Char** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given array member.

- **DDS_ReturnCode_t DDS_DynamicData_get_octet_array** (const **DDS_DynamicData** *self, **DDS_Octet** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given array member.

- **DDS_ReturnCode_t DDS_DynamicData_get_longlong_array** (const **DDS_DynamicData** *self, **DDS_LongLong** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given array member.

- **DDS_ReturnCode_t DDS_DynamicData_get_ulonglong_array** (const **DDS_DynamicData** *self, **DDS_UnsignedLongLong** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given array member.

- **DDS_ReturnCode_t DDS_DynamicData_get_longdouble_array** (const **DDS_DynamicData** *self, **DDS_LongDouble** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given array member.

- **DDS_ReturnCode_t DDS_DynamicData_get_wchar_array** (const **DDS_DynamicData** *self, **DDS_Wchar** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given array member.

- **DDS_ReturnCode_t DDS_DynamicData_get_int8_array** (const **DDS_DynamicData** *self, **DDS_Int8** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given array member.

- **DDS_ReturnCode_t DDS_DynamicData_get_uint8_array** (const **DDS_DynamicData** *self, **DDS_UInt8** *array, **DDS_UnsignedLong** *length, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given array member.

- **DDS_ReturnCode_t DDS_DynamicData_get_long_seq** (const **DDS_DynamicData** *self, struct **DDS_LongSeq** *seq, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_get_short_seq** (const **DDS_DynamicData** *self, struct **DDS_ShortSeq** *seq, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_get_ulong_seq** (const **DDS_DynamicData** *self, struct **DDS_UnsignedLongSeq** *seq, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_get_ushort_seq** (const **DDS_DynamicData** *self, struct **DDS_UnsignedShortSeq** *seq, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_get_float_seq** (const **DDS_DynamicData** *self, struct **DDS_FloatSeq** *seq, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_get_double_seq** (const **DDS_DynamicData** *self, struct **DDS_DoubleSeq** *seq, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_get_boolean_seq** (const **DDS_DynamicData** *self, struct **DDS_BooleanSeq** *seq, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_get_char_seq** (const **DDS_DynamicData** *self, struct **DDS_CharSeq** *seq, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_get_octet_seq** (const **DDS_DynamicData** *self, struct **DDS_OctetSeq** *seq, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_get_longlong_seq** (const **DDS_DynamicData** *self, struct **DDS_LongLongSeq** *seq, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_get_ulonglong_seq** (const **DDS_DynamicData** *self, struct **DDS_UnsignedLongLongSeq** *seq, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_get_longdouble_seq** (const **DDS_DynamicData** *self, struct **DDS_LongDoubleSeq** *seq, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_get_wchar_seq** (const **DDS_DynamicData** *self, struct **DDS_WcharSeq** *seq, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_get_int8_seq** (const **DDS_DynamicData** *self, struct **DDS_Int8Seq** *seq, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_get_uint8_seq** (const **DDS_DynamicData** *self, struct **DDS_UInt8Seq** *seq, const char *member_name, **DDS_DynamicDataMemberId** member_id)

Get a copy of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_set_long** (**DDS_DynamicData** *self, 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. 1203).*

- **DDS_ReturnCode_t DDS_DynamicData_set_short** (**DDS_DynamicData** *self, 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. 1202).*

- **DDS_ReturnCode_t DDS_DynamicData_set_ulong** (**DDS_DynamicData** *self, 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. 1203).*

- **DDS_ReturnCode_t DDS_DynamicData_set_ushort** (**DDS_DynamicData** *self, 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. 1202).*

- **DDS_ReturnCode_t DDS_DynamicData_set_float** (**DDS_DynamicData** *self, 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. 1203).*

- **DDS_ReturnCode_t DDS_DynamicData_set_double** (**DDS_DynamicData** *self, 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. 1203).*

- **DDS_ReturnCode_t DDS_DynamicData_set_boolean** (**DDS_DynamicData** *self, 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. 1204).*

- **DDS_ReturnCode_t DDS_DynamicData_set_char** (**DDS_DynamicData** *self, 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. 1201).*
- **DDS_ReturnCode_t DDS_DynamicData_set_octet** (**DDS_DynamicData** *self, 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. 1202).*
- **DDS_ReturnCode_t DDS_DynamicData_set_longlong** (**DDS_DynamicData** *self, 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. 1203).*
- **DDS_ReturnCode_t DDS_DynamicData_set_ulonglong** (**DDS_DynamicData** *self, 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. 1203).*
- **DDS_ReturnCode_t DDS_DynamicData_set_longdouble** (**DDS_DynamicData** *self, 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. 1204).*
- **DDS_ReturnCode_t DDS_DynamicData_set_wchar** (**DDS_DynamicData** *self, 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. 1202).*
- **DDS_ReturnCode_t DDS_DynamicData_set_int8** (**DDS_DynamicData** *self, 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. 1202).*
- **DDS_ReturnCode_t DDS_DynamicData_set_uint8** (**DDS_DynamicData** *self, 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. 1202).*
- **DDS_ReturnCode_t DDS_DynamicData_set_string** (**DDS_DynamicData** *self, const char *member_name, **DDS_DynamicDataMemberId** member_id, const char *value)
Set the value of the given field of type char.*
- **DDS_ReturnCode_t DDS_DynamicData_set_wstring** (**DDS_DynamicData** *self, const char *member_name, **DDS_DynamicDataMemberId** member_id, const **DDS_Wchar** *value)
*Set the value of the given field of type **DDS_Wchar** (p. 1202)*.*
- **DDS_ReturnCode_t DDS_DynamicData_set_complex_member** (**DDS_DynamicData** *self, const char *member_name, **DDS_DynamicDataMemberId** member_id, const **DDS_DynamicData** *value)
*Copy the state of the given **DDS_DynamicData** (p. 1768) object into a member of this object.*
- **DDS_ReturnCode_t DDS_DynamicData_set_long_array** (**DDS_DynamicData** *self, 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. 1203) or **DDS_Enum** (p. 1204).*
- **DDS_ReturnCode_t DDS_DynamicData_set_short_array** (**DDS_DynamicData** *self, 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 DDS_DynamicData_set_ulong_array** (**DDS_DynamicData** *self, 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. 1203) or **DDS_Enum** (p. 1204).*
- **DDS_ReturnCode_t DDS_DynamicData_set_ushort_array** (**DDS_DynamicData** *self, 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 DDS_DynamicData_set_float_array** (DDS_DynamicData *self, 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 DDS_DynamicData_set_double_array** (DDS_DynamicData *self, 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 DDS_DynamicData_set_boolean_array** (DDS_DynamicData *self, 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 DDS_DynamicData_set_char_array** (DDS_DynamicData *self, 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 DDS_DynamicData_set_octet_array** (DDS_DynamicData *self, 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 DDS_DynamicData_set_longlong_array** (DDS_DynamicData *self, 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 DDS_DynamicData_set_ulonglong_array** (DDS_DynamicData *self, 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 DDS_DynamicData_set_longdouble_array** (DDS_DynamicData *self, 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 DDS_DynamicData_set_wchar_array** (DDS_DynamicData *self, 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 DDS_DynamicData_set_int8_array** (DDS_DynamicData *self, 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 DDS_DynamicData_set_uint8_array** (DDS_DynamicData *self, 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 DDS_DynamicData_set_long_seq** (DDS_DynamicData *self, const char *member_name, DDS_DynamicDataMemberId member_id, const struct DDS_LongSeq *value)
Set the contents of the given sequence member.
- **DDS_ReturnCode_t DDS_DynamicData_set_short_seq** (DDS_DynamicData *self, const char *member_name, DDS_DynamicDataMemberId member_id, const struct DDS_ShortSeq *value)
Set the contents of the given sequence member.
- **DDS_ReturnCode_t DDS_DynamicData_set_ulong_seq** (DDS_DynamicData *self, const char *member_name, DDS_DynamicDataMemberId member_id, const struct DDS_UnsignedLongSeq *value)
Set the contents of the given sequence member.
- **DDS_ReturnCode_t DDS_DynamicData_set_ushort_seq** (DDS_DynamicData *self, const char *member_name, DDS_DynamicDataMemberId member_id, const struct DDS_UnsignedShortSeq *value)

Set the contents of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_set_float_seq** (**DDS_DynamicData** *self, const char *member_name, **DDS_DynamicDataMemberId** member_id, const struct **DDS_FloatSeq** *value)

Set the contents of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_set_double_seq** (**DDS_DynamicData** *self, const char *member_name, **DDS_DynamicDataMemberId** member_id, const struct **DDS_DoubleSeq** *value)

Set the contents of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_set_boolean_seq** (**DDS_DynamicData** *self, const char *member_name, **DDS_DynamicDataMemberId** member_id, const struct **DDS_BooleanSeq** *value)

Set the contents of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_set_char_seq** (**DDS_DynamicData** *self, const char *member_name, **DDS_DynamicDataMemberId** member_id, const struct **DDS_CharSeq** *value)

Set the contents of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_set_octet_seq** (**DDS_DynamicData** *self, const char *member_name, **DDS_DynamicDataMemberId** member_id, const struct **DDS_OctetSeq** *value)

Set the contents of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_set_longlong_seq** (**DDS_DynamicData** *self, const char *member_name, **DDS_DynamicDataMemberId** member_id, const struct **DDS_LongLongSeq** *value)

Set the contents of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_set_ulonglong_seq** (**DDS_DynamicData** *self, const char *member_name, **DDS_DynamicDataMemberId** member_id, const struct **DDS_UnsignedLongLongSeq** *value)

Set the contents of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_set_longdouble_seq** (**DDS_DynamicData** *self, const char *member_name, **DDS_DynamicDataMemberId** member_id, const struct **DDS_LongDoubleSeq** *value)

Set the contents of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_set_wchar_seq** (**DDS_DynamicData** *self, const char *member_name, **DDS_DynamicDataMemberId** member_id, const struct **DDS_WcharSeq** *value)

Set the contents of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_set_int8_seq** (**DDS_DynamicData** *self, const char *member_name, **DDS_DynamicDataMemberId** member_id, const struct **DDS_Int8Seq** *value)

Set the contents of the given sequence member.

- **DDS_ReturnCode_t DDS_DynamicData_set_uint8_seq** (**DDS_DynamicData** *self, const char *member_name, **DDS_DynamicDataMemberId** member_id, const struct **DDS_UInt8Seq** *value)

Set the contents of the given sequence member.

- struct **DDS_DynamicDataTypeSupport** * **DDS_DynamicDataTypeSupport_new** (const **DDS_TypeCode** *type, const struct **DDS_DynamicDataTypeProperty_t** *props)

*Construct a new **DDS_DynamicDataTypeSupport** (p. 326) object.*

- void **DDS_DynamicDataTypeSupport_delete** (struct **DDS_DynamicDataTypeSupport** *self)

*Delete a **DDS_DynamicDataTypeSupport** (p. 326) object.*

- **DDS_ReturnCode_t DDS_DynamicDataTypeSupport_register_type** (struct **DDS_DynamicDataTypeSupport** *self, **DDS_DomainParticipant** *participant, const char *type_name)

*Associate the **DDS_TypeCode** (p. 2071) with the given **DDS_DomainParticipant** (p. 77) under the given logical name.*

- **DDS_ReturnCode_t DDS_DynamicDataTypeSupport_unregister_type** (struct **DDS_DynamicDataTypeSupport** *self, **DDS_DomainParticipant** *participant, const char *type_name)

*Remove the definition of this type from the **DDS_DomainParticipant** (p. 77).*

- const char * **DDS_DynamicDataTypeSupport_get_type_name** (const struct **DDS_DynamicDataTypeSupport** *self)

Get the default name of this type.

- `const struct DDS_TypeCode * DDS_DynamicDataTypeSupport_get_data_type` (`const struct DDS_DynamicDataTypeSupport *self`)
*Get the **DDS_TypeCode** (p. 2071) wrapped by this **DDS_DynamicDataTypeSupport** (p. 326).*
- `DDS_DynamicData * DDS_DynamicDataTypeSupport_create_data` (`struct DDS_DynamicDataTypeSupport *self`)
*Create a new **DDS_DynamicData** (p. 1768) sample initialized with the **DDS_TypeCode** (p. 2071) and properties of this **DDS_DynamicDataTypeSupport** (p. 326).*
- `DDS_ReturnCode_t DDS_DynamicDataTypeSupport_delete_data` (`struct DDS_DynamicDataTypeSupport *self, DDS_DynamicData *a_data`)
*Finalize and deallocate the **DDS_DynamicData** (p. 1768) sample.*
- `void DDS_DynamicDataTypeSupport_print_data` (`const struct DDS_DynamicDataTypeSupport *self, const DDS_DynamicData *a_data`)
Print a string representation of the given sample to the given file.
- `DDS_ReturnCode_t DDS_DynamicDataTypeSupport_copy_data` (`const struct DDS_DynamicDataTypeSupport *self, DDS_DynamicData *dest, const DDS_DynamicData *source`)
Deeply copy the given data samples.
- `DDS_ReturnCode_t DDS_DynamicDataTypeSupport_initialize_data` (`const struct DDS_DynamicDataTypeSupport *self, DDS_DynamicData *a_data`)
*Initialize a **DDS_DynamicData** (p. 1768) object to a valid empty state.*
- `DDS_ReturnCode_t DDS_DynamicDataTypeSupport_finalize_data` (`const struct DDS_DynamicDataTypeSupport *self, DDS_DynamicData *a_data`)
Clean up the internal state of this object to get it ready for deallocation, but don't deallocate the pointer.

Variables

- `const struct DDS_DynamicDataProperty_t DDS_DYNAMIC_DATA_PROPERTY_DEFAULT`
*Sentinel constant indicating default values for **DDS_DynamicDataProperty_t** (p. 1779).*
- `const struct DDS_DynamicDataTypeProperty_t DDS_DYNAMIC_DATA_TYPE_PROPERTY_DEFAULT`
*Sentinel constant indicating default values for **DDS_DynamicDataTypeProperty_t** (p. 1781).*
- `struct DDS_DynamicDataJsonParserProperties_t DDS_DYNAMIC_DATA_JSON_PARSER_PROPERTIES_DEFAULT`
*Sentinel constant indicating default values for **DDS_DynamicDataProperty_t** (p. 1779).*

5.14.1 Detailed Description

<<**extension**>> (p. 1012) 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. 2071) (see **Type Code Support** (p. 228)) that defines the type definition you want to use.**

A **DDS_TypeCode** (p. 2071) includes a type's *kind* (**DDS_TCKind** (p. 243)), *name*, and *members* (that is, fields). You can create your own **DDS_TypeCode** (p. 2071) using the **DDS_TypeCodeFactory** (p. 2071) class – see, for example, the **DDS_TypeCodeFactory_create_struct_tc** (p. 291) method. Alternatively, you can use a remote **DDS_TypeCode** (p. 2071) that you discovered on the network (see **Built-in Topics** (p. 165)) or one generated by `rtiddsgen` (see the *Code Generator User's Manual*).

2. Wrap the `DDS_TypeCode` (p. 2071) in a `DDS_DynamicDataSupport` (p. 326) object.

See the constructor `DDS_DynamicDataSupport_new` (p. 432). This object lets you connect the type definition to a `DDS_DomainParticipant` (p. 77) and manage data samples (of type `DDS_DynamicData` (p. 1768)).

3. Register the `DDS_DynamicDataSupport` (p. 326) with one or more domain participants.

See `DDS_DynamicDataSupport_register_type` (p. 433). 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 `DDS_Topic` (p. 175) from the `DDS_DomainParticipant` (p. 77).

Use the name under which you registered your data type – see `DDS_DomainParticipant_create_topic` (p. 115). This `DDS_Topic` (p. 175) is what you will use to produce and consume data.

5. Create a `DDS_DynamicDataWriter` (p. 325) and/or `DDS_DynamicDataReader` (p. 326).

These objects will produce and/or consume data (of type `DDS_DynamicData` (p. 1768)) on the `DDS_Topic` (p. 175). You can create these objects directly from the `DDS_DomainParticipant` (p. 77) – see `DDS_DomainParticipant_create_datawriter` (p. 108) and `DDS_DomainParticipant_create_datareader` (p. 112) – or by first creating intermediate `DDS_Publisher` (p. 443) and `DDS_Subscriber` (p. 572) objects – see `DDS_DomainParticipant_create_publisher` (p. 103) and `DDS_DomainParticipant_create_subscriber` (p. 106).

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 `DDS_DomainParticipant` (p. 77) is optional; all types are automatically unregistered when the `DDS_DomainParticipant` (p. 77) itself is deleted.

5.14.2 Macro Definition Documentation

5.14.2.1 `DDS_DynamicDataProperty_t_INITIALIZER`

```
#define DDS_DynamicDataProperty_t_INITIALIZER
```

Static initializer for `DDS_DynamicDataProperty_t` (p. 1779) objects.

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

```
struct DDS_DynamicDataProperty_t props = DDS_DynamicDataProperty_t_INITIALIZER;
```

See also

`DDS_DynamicDataProperty_t` (p. 1779)

5.14.2.2 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. 325)

5.14.2.3 DDS_DynamicDataTypeProperty_t_INITIALIZER

```
#define DDS_DynamicDataTypeProperty_t_INITIALIZER
```

Static initializer for **DDS_DynamicDataTypeProperty_t** (p. 1781) objects.

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

```
struct DDS_DynamicDataTypeProperty_t props =  
    DDS_DynamicDataTypeProperty_t_INITIALIZER;
```

See also

DDS_DynamicDataTypeProperty_t (p. 1781)

5.14.3 Typedef Documentation

5.14.3.1 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. 324)

5.14.3.2 DDS_DynamicDataWriter

```
typedef struct DDS_DynamicDataWriter DDS_DynamicDataWriter
```

Writes (publishes) objects of type **DDS_DynamicData** (p. 1768).

Instantiates **DDS_DataWriter** (p. 485) < **DDS_DynamicData** (p. 1768) > .

See also

DDS_DataWriter (p. 485)

FooDataWriter (p. 2113)

DDS_DynamicData (p. 1768)

5.14.3.3 DDS_DynamicDataReader

```
typedef struct DDS_DynamicDataReader DDS_DynamicDataReader
```

Reads (subscribes to) objects of type **DDS_DynamicData** (p. 1768).

Instantiates **DDS_DataReader** (p. 615) < **DDS_DynamicData** (p. 1768) > .

See also

DDS_DataReader (p. 615)

FooDataReader (p. 2112)

DDS_DynamicData (p. 1768)

5.14.3.4 DDS_DynamicDataTypeSupport

```
typedef struct DDS_DynamicDataTypeSupport DDS_DynamicDataTypeSupport
```

A factory for registering a dynamically defined type and creating **DDS_DynamicData** (p. 1768) objects.

A **DDS_DynamicDataTypeSupport** (p. 326) has three roles:

1. It associates a **DDS_TypeCode** (p. 2071) with policies for managing objects of that type. See the constructor, **DDS_DynamicDataTypeSupport_new** (p. 432).
2. It registers its type under logical names with a **DDS_DomainParticipant** (p. 77). See **DDS_DynamicDataTypeSupport_register_type** (p. 433).
3. It creates **DDS_DynamicData** (p. 1768) samples pre-initialized with the type and properties of the type support itself. See **DDS_DynamicDataTypeSupport_create_data** (p. 435).

5.14.4 Function Documentation

5.14.4.1 DDS_DynamicData_initialize()

```
DDS_Boolean DDS_DynamicData_initialize (
    DDS_DynamicData * self,
    const DDS_TypeCode * type,
    const struct DDS_DynamicDataProperty_t * property )
```

Initialize a **DDS_DynamicData** (p. 1768) object to a valid empty state.

Use the constructor, **DDS_DynamicData_new** (p. 327), to allocate a new **DDS_DynamicData** (p. 1768) object on the heap. Use this method to initialize a **DDS_DynamicData** (p. 1768) object that is stored deeply, either within another object or on the stack.

When you're finished with the object, dispose of it with **DDS_DynamicData_finalize** (p. 328).

```
struct DDS_DynamicData sample;

DDS_Boolean succeeded = DDS_DynamicData_initialize(
    &sample, myType, myProperties);

/* Handle failure... */
/* Do something... */

DDS_DynamicData_finalize(&sample);
```

MT Safety:

UNSAFE.

See also

DDS_DynamicData_finalize (p. 328)

DDS_DynamicData_new (p. 327)

5.14.4.2 DDS_DynamicData_new()

```
DDS_DynamicData * DDS_DynamicData_new (
    const DDS_TypeCode * type,
    const struct DDS_DynamicDataProperty_t * property )
```

The constructor for new **DDS_DynamicData** (p. 1768) objects.

This method returns a fully initialized object. It is *not* necessary to subsequently call **DDS_DynamicData_initialize** (p. 327).

If initialization fails, this method returns NULL.

The type parameter may be NULL. In that case, this **DDS_DynamicData** (p. 1768) must be *bound* with **DDS_DynamicData_bind_type** (p. 340) or **DDS_DynamicData_bind_complex_member** (p. 342) before it can be used.

If the **DDS_TypeCode** (p. 2071) is not NULL, the newly constructed **DDS_DynamicData** (p. 1768) object will retain a reference to it. It is *not* safe to delete the **DDS_TypeCode** (p. 2071) until all samples that use it have themselves been deleted.

In most cases, it is not necessary to call this constructor explicitly. Instead, use **DDS_DynamicData_TypeSupport_create_data** (p. 435), and the **DDS_TypeCode** (p. 2071) 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 **DDS_DynamicData_TypeSupport** (p. 326) object maintains the **DDS_TypeCode** (p. 2071) used by the samples it creates.

However you create a **DDS_DynamicData** (p. 1768) 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_delete** (p. 329).

```
struct DDS_DynamicData* sample = DDS_DynamicData_new(
    myType, myProperties);

/* Failure indicated by NULL return result. */
/* Do something... */

DDS_DynamicData_delete(sample);
```

Parameters

<i>type</i>	<< <i>in</i> >> (p. 1013) The type of which the new object will represent an object.
<i>property</i>	<< <i>in</i> >> (p. 1013) Properties that configure the behavior of the new object. Most users can simply use DDS_DYNAMIC_DATA_PROPERTY_DEFAULT (p. 437).

See also

DDS_DynamicData_initialize (p. 327)
DDS_DynamicData_delete (p. 329)
DDS_DynamicData_TypeSupport_create_data (p. 435)

Examples

HelloWorldPlugin.c.

5.14.4.3 DDS_DynamicData_finalize()

```
void DDS_DynamicData_finalize (
    DDS_DynamicData * self )
```

Clean up the internal state of this object to get it ready for deallocation, but don't deallocate the pointer.

Use **DDS_DynamicData_delete** (p. 329) to clean up a **DDS_DynamicData** (p. 1768) object on the heap. Use this method to clean up a **DDS_DynamicData** (p. 1768) object that is stored deeply, either within another object or on the stack.

MT Safety:

UNSAFE.

See also

DDS_DynamicData_delete (p. 329)

DDS_DynamicData_initialize (p. 327)

5.14.4.4 DDS_DynamicData_delete()

```
void DDS_DynamicData_delete (
    DDS_DynamicData * self )
```

Finalize and deallocate this **DDS_DynamicData** (p. 1768) sample.

This method fully finalizes the object. It is *not* necessary to previously call **DDS_DynamicData_finalize** (p. 328).

MT Safety:

UNSAFE.

See also

DDS_DynamicData_finalize (p. 328)

DDS_DynamicData_new (p. 327)

Examples

HelloWorldPlugin.c.

5.14.4.5 DDS_DynamicData_copy()

```
DDS_ReturnCode_t DDS_DynamicData_copy (
    DDS_DynamicData * self,
    const DDS_DynamicData * src )
```

Deeply copy from the given object to this object.

MT Safety:

UNSAFE.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↔PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

5.14.4.6 DDS_DynamicData_equal()

```
DDS_Boolean DDS_DynamicData_equal (
    const DDS_DynamicData * self,
    const DDS_DynamicData * other )
```

Indicate whether the contents of another **DDS_DynamicData** (p. 1768) 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. 1201). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

5.14.4.7 DDS_DynamicData_clear_all_members()

```
DDS_ReturnCode_t DDS_DynamicData_clear_all_members (
    DDS_DynamicData * self )
```

Clear the contents of all data members of this object.

MT Safety:

UNSAFE.

Returns

One of the **Standard Return Codes** (p. 1221)

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

See also

DDS_DynamicData_clear_optional_member (p. 330)

5.14.4.8 DDS_DynamicData_clear_optional_member()

```
DDS_ReturnCode_t DDS_DynamicData_clear_optional_member (
    DDS_DynamicData * self,
    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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member (to look up the member by name), or NULL (to look up the member by its ID).
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member (to look up the member by its ID), or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) (to look up the member by name). See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DynamicData_clear_all_members (p. 330)

Clear an optional field

5.14.4.9 DDS_DynamicData_clear_member()

```
DDS_ReturnCode_t DDS_DynamicData_clear_member (
    DDS_DynamicData * self,
    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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member to clear or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member to clear or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DynamicData_clear_all_members (p. 330)

Clear a field

5.14.4.10 DDS_DynamicData_set_cdr_buffer()

```
DDS_ReturnCode_t DDS_DynamicData_set_cdr_buffer (
    DDS_DynamicData * self,
    const char * buffer,
    DDS_UnsignedLong length )
```

Associate a CDR buffer with a DynamicData object.

This function 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 function will return an error.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>buffer</i>	<< <i>in</i> >> (p. 1013) CDR buffer.
<i>length</i>	<< <i>in</i> >> (p. 1013) CDR buffer length.

Returns

One of the **Standard Return Codes** (p. 1221)

5.14.4.11 DDS_DynamicData_get_cdr_buffer()

```
const char * DDS_DynamicData_get_cdr_buffer (
    const DDS_DynamicData * self,
    DDS_UnsignedLong * length )
```

Get the CDR buffer associated with a DynamicData object.

This function returns the CDR buffer associated with a DynamicData object.

If the DynamicData object is not using CDR as its data representation, this function 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. 1783) to **DDS_BOOLEAN_TRUE** (p. 1201).

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>length</i>	<< <i>out</i> >> (p. 1013) CDR buffer length.

Returns

One of the **Standard Return Codes** (p. 1221)

5.14.4.12 DDS_DynamicData_is_cdr()

```
DDS_Boolean DDS_DynamicData_is_cdr (
    const DDS_DynamicData * self )
```

Determine if a DynamicData object is using CDR as its data representation.

This function returns **DDS_BOOLEAN_TRUE** (p. 1201) if the DynamicData object is using CDR as its data representation, and **DDS_BOOLEAN_FALSE** (p. 1201) 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_DynamicDataTypeSerializationProperty_t::skip_deserialization** (p. 1783) has been set to **DDS_BOOLEAN_TRUE** (p. 1201) or **DDS_DynamicData_set_cdr_buffer** (p. 332) has been used to associate a CDR buffer with the DynamicData object. See each function's documentation for whether or not the function is supported for DynamicData objects in CDR format.

See also

DDS_DynamicData_get_cdr_buffer (p. 332)

DDS_DynamicData_set_cdr_buffer (p. 332)

DDS_DynamicDataTypeSerializationProperty_t::skip_deserialization (p. 1783)

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.14.4.13 DDS_DynamicData_set_discriminator()

```
DDS_ReturnCode_t DDS_DynamicData_set_discriminator (
    DDS_DynamicData * self,
    DDS_Long discriminator )
```

Set the value of the union discriminator (UnionType only)

This operation is valid only for objects of **DDS_TCKind** (p. 243) **DDS_TK_UNION** (p. 243).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>discriminator</i>	<< <i>in</i> >> (p. 1013) The discriminator value to set

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DynamicData_get_discriminator (p. 335)

Getting and Setting Union Discriminator Values (p. 1769)

DDS_DynamicData_get_member_count (p. 345)

5.14.4.14 DDS_DynamicData_get_discriminator()

```
DDS_ReturnCode_t DDS_DynamicData_get_discriminator (
    const DDS_DynamicData * self,
    DDS_Long * discriminator )
```

Get the current value of the union discriminator.

This operation is valid only for objects of **DDS_TCKind** (p. 243) **DDS_TK_UNION** (p. 243).

This API should be used instead of **DDS_DynamicData_get_member_info_by_index** (p. 348), 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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>discriminator</i>	<< <i>out</i> >> (p. 1013) The current value of the union discriminator.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DynamicData_set_discriminator (p. 334)

Getting and Setting Union Discriminator Values (p. 1769)

5.14.4.15 DDS_DynamicData_print()

```
DDS_ReturnCode_t DDS_DynamicData_print (
    const DDS_DynamicData * self,
    FILE * fp,
    int indent )
```

Output a textual representation of this object and its contents to the given file.

This method is equivalent to **DDS_DynamicDataTypeSupport_print_data** (p. 435).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵ PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

Precondition

The type must be an aggregation or collection. That is, its **DDS_TCKind** (p. 243) must be one of: **DDS_TK_↵ STRUCT** (p. 243), **DDS_TK_VALUE** (p. 243), **DDS_TK_UNION** (p. 243), **DDS_TK_SEQUENCE** (p. 243) or **DDS_↵ TK_ARRAY** (p. 243), otherwise this function fails with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>fp</i>	<< <i>in</i> >> (p. 1013) The file into which the object should be printed (to print to standard output, provide the stream pointer 'stdout')
<i>indent</i>	<< <i>in</i> >> (p. 1013) The output of this method will be pretty-printed. This argument indicates the amount of initial indentation of the output.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DynamicDataSupport_print_data (p. 435)

5.14.4.16 DDS_DynamicData_from_cdr_buffer()

```
DDS_ReturnCode_t DDS_DynamicData_from_cdr_buffer (
    DDS_DynamicData * self,
    const char * buffer,
    unsigned int length )
```

Deserializes a DynamicData object from a buffer of octets.

This function deserializes a DynamicData object from a CDR buffer of octets.

The content of the buffer generated by the function **DDS_DynamicData_to_cdr_buffer** (p. 338) can be provided to this function 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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>buffer</i>	<< <i>in</i> >> (p. 1013). Deserialization buffer.
<i>length</i>	<< <i>in</i> >> (p. 1013). Length of the serialized representation of the DynamicData object in the buffer.

Returns

One of the **Standard Return Codes** (p. 1221)

Examples

HelloWorldPlugin.c.

5.14.4.17 DDS_DynamicData_to_cdr_buffer_ex()

```
DDS_ReturnCode_t DDS_DynamicData_to_cdr_buffer_ex (
    DDS_DynamicData * self,
    char * buffer,
    unsigned int * length,
    DDS_DataRepresentationId_t representation )
```

Serializes a DynamicData object into a buffer of octets.

This function serializes a DynamicData object into a buffer of octets using the input data representation. See **DDS_DynamicData_to_cdr_buffer** (p. 338) for details.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>buffer</i>	<< <i>out</i> >> (p. 1013). Serialization buffer.
<i>length</i>	<< <i>inout</i> >> (p. 1013). Serialization buffer length.
<i>representation</i>	<< <i>in</i> >> (p. 1013). Representation used to serialize the data

5.14.4.18 DDS_DynamicData_to_cdr_buffer()

```
DDS_ReturnCode_t DDS_DynamicData_to_cdr_buffer (
    DDS_DynamicData * self,
    char * buffer,
    unsigned int * length )
```

Serializes a DynamicData object into a CDR buffer of octets.

This function serializes a DynamicData object into a buffer of octets, using CDR as the data representation. Calling this function is equivalent to calling **DDS_DynamicData_to_cdr_buffer_ex** (p. 338) with **DDS_AUTO_DATA_REPRESENTATION** (p. 1258) as the representation.

The input buffer must be large enough to store the serialized representation of the DynamicData object. Otherwise, the function 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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>buffer</i>	<< <i>out</i> >> (p. 1013). Serialization buffer.
<i>length</i>	<< <i>inout</i> >> (p. 1013). When the buffer is set to NULL, after the function 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 function is invoked. After the function 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.

Returns

One of the **Standard Return Codes** (p. 1221)

5.14.4.19 DDS_DynamicData_to_string()

```
DDS_ReturnCode_t DDS_DynamicData_to_string (
    DDS_DynamicData * self,
    char * str,
    DDS_UnsignedLong * str_size,
    const struct DDS_PrintFormatProperty * property )
```

Get a string representation of a DynamicData object.

This function 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 function 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. 1222).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵ PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>str</i>	<< <i>out</i> >> (p. 1013). Output string representing the dynamic data sample.
<i>str_size</i>	<< <i>inout</i> >> (p. 1013). When <i>str</i> is set to NULL, after the function executes, <i>str_size</i> will contain a buffer size big enough to hold the string representation of the data. When <i>str</i> is not NULL, <i>str_size</i> must contain the size of the input buffer when the function is invoked. If the size of the input buffer is too small, after the function executes, <i>str_size</i> will be updated to contain the required size of the string content and the function will return DDS_RETCODE_OUT_OF_RESOURCES (p. 1222).
<i>property</i>	<< <i>in</i> >> (p. 1013). Properties describing what the format of the output string should be. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

5.14.4.20 DDS_DynamicData_from_string()

```
DDS_ReturnCode_t DDS_DynamicData_from_string (
    DDS_DynamicData * self,
    const char * str,
    DDS_PrintFormatKind format_kind )
```

Populates a DynamicData object from a JSON string representation.

This function takes a JSON string representing a data sample of this type, and uses it to populate this DynamicData.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>str</i>	<< <i>in</i> >> (p. 1013). The JSON string representation
<i>format_kind</i>	<< <i>in</i> >> (p. 1013). Must be set to DDS_JSON_PRINT_FORMAT (p. 186), the only currently supported format.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

5.14.4.21 DDS_DynamicData_get_info()

```
void DDS_DynamicData_get_info (
    const DDS_DynamicData * self,
    struct DDS_DynamicDataInfo * info_out )
```

Fill in the given descriptor with information about this **DDS_DynamicData** (p. 1768).

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. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>info_out</i>	<< <i>out</i> >> (p. 1013) The descriptor object whose contents will be overwritten by this operation.

5.14.4.22 DDS_DynamicData_bind_type()

```
DDS_ReturnCode_t DDS_DynamicData_bind_type (
    DDS_DynamicData * self,
    const DDS_TypeCode * type )
```

If this **DDS_DynamicData** (p. 1768) object is not yet associated with a data type, set that type now to the given **DDS_TypeCode** (p. 2071).

This advanced operation allows you to reuse a single **DDS_DynamicData** (p. 1768) 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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

```
struct DDS_DynamicData* myData = DDS_DynamicData_new(NULL, myProperties);
```

```
DDS_TypeCode* myType = ...;
```

```
DDS_DynamicData_bind_type(myData, myType);
```

```
/* Do something... */
```

```
DDS_DynamicData_unbind_type(myData);
```

```
DDS_DynamicData_delete(myData);
```

Note that the **DDS_DynamicData** (p. 1768) object will retain a reference to the **DDS_TypeCode** (p. 2071) object you provide. It is *not* safe to delete the **DDS_TypeCode** (p. 2071) until after it is unbound.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>type</i>	<< <i>in</i> >> (p. 1013) The type to associate with this DDS_DynamicData (p. 1768) object.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DynamicData_unbind_type (p. 341)

5.14.4.23 DDS_DynamicData_unbind_type()

```
DDS_ReturnCode_t DDS_DynamicData_unbind_type (
    DDS_DynamicData * self )
```

Dissociate this **DDS_DynamicData** (p. 1768) 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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DynamicData_bind_type (p. 340)

DDS_DynamicData_clear_all_members (p. 330)

5.14.4.24 DDS_DynamicData_bind_complex_member()

```
DDS_ReturnCode_t DDS_DynamicData_bind_complex_member (
    DDS_DynamicData * self,
    DDS_DynamicData * value_out,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Use another **DDS_DynamicData** (p. 1768) object to provide access to a complex field of this **DDS_DynamicData** (p. 1768) object.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵**
PRECONDITION_NOT_MET (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) 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. 1768) object to that member. This operation will bind the type code of the member to the provided **DDS_DynamicData** (p. 1768) object and perform additional initialization.

The following example demonstrates the usage pattern. Note that error handling has been omitted for brevity.

```
struct DDS_DynamicData* outer = ...;

float theFloatValue = 0;

struct DDS_DynamicData* toBeBound = DDS_DynamicData_new(NULL, myProperties);

DDS_DynamicData_bind_complex_member(
    outer,
    toBeBound,
    "complexMember",
    DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED);

DDS_DynamicData_get_float(
    toBeBound,
    &theFloat,
    "theFloat"
    DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED);

DDS_DynamicData_unbind_complex_member(outer, 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 constructor and is not provided with a **DDS_TypeCode** (p. 2071).

Only a single member of a given **DDS_DynamicData** (p. 1768) 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. 344) 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. 365) 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. 365) 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. 1771).

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>value_out</i>	<< <i>out</i> >> (p. 1013) The object that you wish to bind to the field.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DynamicData_unbind_complex_member (p. 344)

DDS_DynamicData_get_complex_member (p. 365)

5.14.4.25 DDS_DynamicData_unbind_complex_member()

```
DDS_ReturnCode_t DDS_DynamicData_unbind_complex_member (
    DDS_DynamicData * self,
    DDS_DynamicData * value )
```

Tear down the association created by a **DDS_DynamicData_bind_complex_member** (p. 342) 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. 342) 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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>value</i>	<< <i>in</i> >> (p. 1013) The same object you passed to DDS_DynamicData_bind_complex_member (p. 342). This argument is used for error checking purposes.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_bind_complex_member (p. 342)

5.14.4.26 DDS_DynamicData_get_type()

```
const DDS_TypeCode * DDS_DynamicData_get_type (
    const DDS_DynamicData * self )
```

Get the data type, of which this **DDS_DynamicData** (p. 1768) represents an instance.

MT Safety:

UNSAFE.

5.14.4.27 DDS_DynamicData_get_type_kind()

```
DDS_TCKind DDS_DynamicData_get_type_kind (
    const DDS_DynamicData * self )
```

Get the kind of this object's data type.

This is a convenience method. It's equivalent to calling **DDS_DynamicData_get_type** (p. 344) followed by **DDS_TypeCode_kind** (p. 244).

MT Safety:

UNSAFE.

5.14.4.28 DDS_DynamicData_get_member_count()

```
DDS_UnsignedLong DDS_DynamicData_get_member_count (
    const DDS_DynamicData * self )
```

Get the number of members in the type.

For objects of type kind **DDS_TK_ARRAY** (p. 243) or **DDS_TK_SEQUENCE** (p. 243), this method returns the number of elements in the collection.

For objects of type kind **DDS_TK_STRUCT** (p. 243) or **DDS_TK_VALUE** (p. 243), it returns the number of fields in the type.

For objects of type kind **DDS_TK_UNION** (p. 243), 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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

See also

DDS_DynamicData_get_member_info_by_index (p. 348)

5.14.4.29 DDS_DynamicData_member_exists()

```
DDS_Boolean DDS_DynamicData_member_exists (
    const DDS_DynamicData * self,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

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. 1778), which is retrieved with **DDS_DynamicData_get_member_info** (p. 347).

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. 1201). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<<in>> (p. 1013) Cannot be NULL.
<i>member_name</i>	<<in>> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<<in>> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

See also

DDS_DynamicData_member_exists_in_type (p. 346)

DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324)

5.14.4.30 DDS_DynamicData_member_exists_in_type()

```
DDS_Boolean DDS_DynamicData_member_exists_in_type (
    const DDS_DynamicData * self,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

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. 1201). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

See also

DDS_DynamicData_member_exists (p. 345)

DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324)

5.14.4.31 DDS_DynamicData_get_member_info()

```
DDS_ReturnCode_t DDS_DynamicData_get_member_info (
    const DDS_DynamicData * self,
    struct DDS_DynamicDataMemberInfo * info,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Fill in the given descriptor with information about the identified member of this **DDS_DynamicData** (p. 1768) sample.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↔PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

This operation is valid for objects of **DDS_TCKind** (p. 243) **DDS_TK_ARRAY** (p. 243), **DDS_TK_SEQUENCE** (p. 243), **DDS_TK_STRUCT** (p. 243), **DDS_TK_UNION** (p. 243), and **DDS_TK_VALUE** (p. 243).

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. 1222). 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. 1778) set to false.
- If the specified member is defined in the type and exists in this data sample, **DDS_DynamicDataMemberInfo_↔::member_exists** (p. 1778) 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. 1222).
- 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. 1778) set to false.
- If `member_id` is smaller than or equal to the current length, **DDS_DynamicDataMemberInfo::member_exists** (p. 1778) is true.

When this sample represents an array, this function either fails with **DDS_RETCODE_NO_DATA** (p. 1222) when the index is out of bounds or else returns an object with **DDS_DynamicDataMemberInfo::member_exists** (p. 1778) set to true.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>info</i>	<< <i>out</i> >> (p. 1013) The descriptor object whose contents will be overwritten by this operations.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) 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.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member for which to get the info, or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DynamicData_get_member_info_by_index (p. 348)

DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324)

5.14.4.32 DDS_DynamicData_get_member_info_by_index()

```
DDS_ReturnCode_t DDS_DynamicData_get_member_info_by_index (
    const DDS_DynamicData * self,
    struct DDS_DynamicDataMemberInfo * info,
    DDS_UnsignedLong index )
```

Fill in the given descriptor with information about the identified member of this **DDS_DynamicData** (p. 1768) sample.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

This operation is valid for objects of **DDS_TCKind** (p. 243) **DDS_TK_ARRAY** (p. 243), **DDS_TK_SEQUENCE** (p. 243), **DDS_TK_STRUCT** (p. 243), **DDS_TK_VALUE** (p. 243), and **DDS_TK_UNION** (p. 243).

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. 335) 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. 1222).

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>info</i>	<< <i>out</i> >> (p. 1013) The descriptor object whose contents will be overwritten by this operations.
<i>index</i>	<< <i>in</i> >> (p. 1013) The zero-base of the member for which to get the info.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DynamicData_get_member_info (p. 347)

DDS_DynamicData_get_member_count (p. 345)

DDS_DynamicData_get_discriminator (p. 335)

DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324)

5.14.4.33 DDS_DynamicData_get_member_type()

```
DDS_ReturnCode_t DDS_DynamicData_get_member_type (
    const   DDS_DynamicData * self,
    const   DDS_TypeCode ** type_out,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

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. 243) **DDS_TK_ARRAY** (p. 243), **DDS_TK_SEQUENCE** (p. 243), **DDS_TK_STRUCT** (p. 243), and **DDS_TK_VALUE** (p. 243). For type kinds **DDS_TK_ARRAY** (p. 243) and **DDS_TK_SEQUENCE** (p. 243), 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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>type_out</i>	<< <i>out</i> >> (p. 1013) If this method returned success, this argument refers to the found member's type. It cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DynamicData_get_member_info (p. 347)

DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324)

5.14.4.34 DDS_DynamicData_is_member_key()

```
DDS_ReturnCode_t DDS_DynamicData_is_member_key (
    const DDS_DynamicData * self,
    DDS_Boolean * is_key_out,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

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. 243) or **DDS_TK_VALUE** (p. 243).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↔PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>is_key_out</i>	<< <i>out</i> >> (p. 1013) If this method returned success, this argument indicates whether the indicated member is part of the key. It cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221)

5.14.4.35 DDS_DynamicData_get_long()

```
DDS_ReturnCode_t DDS_DynamicData_get_long (
    const DDS_DynamicData * self,
    DDS_Long * value_out,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get the value of the given field, which is of type **DDS_Long** (p. 1203) or another type implicitly convertible to it (**DDS_↔Octet** (p. 1202), **DDS_Char** (p. 1201), **DDS_Short** (p. 1202), **DDS_UnsignedShort** (p. 1202), or **DDS_Enum** (p. 1204)).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>value_out</i>	<< <i>out</i> >> (p. 1013) If this method returned success, this argument will contain the value of the indicated member. It cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_long (p. 394)

5.14.4.36 DDS_DynamicData_get_short()

```
DDS_ReturnCode_t DDS_DynamicData_get_short (
    const DDS_DynamicData * self,
    DDS_Short * value_out,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get the value of the given field, which is of type **DDS_Short** (p. 1202) or another type implicitly convertible to it (**DDS_Octet** (p. 1202) or **DDS_Char** (p. 1201)).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>value_out</i>	<< <i>out</i> >> (p. 1013) If this method returned success, this argument will contain the value of the indicated member. It cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_short (p. 394)

5.14.4.37 DDS_DynamicData_get_ulong()

```
DDS_ReturnCode_t DDS_DynamicData_get_ulong (
    const DDS_DynamicData * self,
    DDS_UnsignedLong * value_out,
```



```
const char * member_name,
DDS_DynamicDataMemberId member_id )
```

Get the value of the given field, which is of type **DDS_UnsignedLong** (p. 1203) or another type implicitly convertible to it (**DDS_Octet** (p. 1202), **DDS_Char** (p. 1201), **DDS_Short** (p. 1202), **DDS_UnsignedShort** (p. 1202), or **DDS_Enum** (p. 1204)).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>value_out</i>	<< <i>out</i> >> (p. 1013) If this method returned success, this argument will contain the value of the indicated member. It cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_ulong (p. 395)

5.14.4.38 DDS_DynamicData_get_ushort()

```
DDS_ReturnCode_t DDS_DynamicData_get_ushort (
    const DDS_DynamicData * self,
    DDS_UnsignedShort * value_out,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get the value of the given field, which is of type **DDS_UnsignedShort** (p. 1202) or another type implicitly convertible to it (**DDS_Octet** (p. 1202) or **DDS_Char** (p. 1201)).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>value_out</i>	<< <i>out</i> >> (p. 1013) If this method returned success, this argument will contain the value of the indicated member. It cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_ushort (p. 396)

5.14.4.39 DDS_DynamicData_get_float()

```
DDS_ReturnCode_t DDS_DynamicData_get_float (
    const DDS_DynamicData * self,
    DDS_Float * value_out,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get the value of the given field, which is of type **DDS_Float** (p. 1203).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>value_out</i>	<< <i>out</i> >> (p. 1013) If this method returned success, this argument will contain the value of the indicated member. It cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_float (p. 396)

5.14.4.40 DDS_DynamicData_get_double()

```
DDS_ReturnCode_t DDS_DynamicData_get_double (
    const DDS_DynamicData * self,
    DDS_Double * value_out,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get the value of the given field, which is of type **DDS_Double** (p. 1203) or another type implicitly convertible to it (**DDS_↵_Float** (p. 1203)).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>value_out</i>	<< <i>out</i> >> (p. 1013) If this method returned success, this argument will contain the value of the indicated member. It cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_double (p. 397)

5.14.4.41 DDS_DynamicData_get_boolean()

```
DDS_ReturnCode_t DDS_DynamicData_get_boolean (
    const DDS_DynamicData * self,
    DDS_Boolean * value_out,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get the value of the given field, which is of type **DDS_Boolean** (p. 1204).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>value_out</i>	<< <i>out</i> >> (p. 1013) If this method returned success, this argument will contain the value of the indicated member. It cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_boolean (p. 398)

5.14.4.42 DDS_DynamicData_get_char()

```
DDS_ReturnCode_t DDS_DynamicData_get_char (
    const DDS_DynamicData * self,
    DDS_Char * value_out,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get the value of the given field, which is of type **DDS_Char** (p. 1201).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>value_out</i>	<< <i>out</i> >> (p. 1013) If this method returned success, this argument will contain the value of the indicated member. It cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_char (p. 399)

5.14.4.43 DDS_DynamicData_get_octet()

```
DDS_ReturnCode_t DDS_DynamicData_get_octet (
    const DDS_DynamicData * self,
    DDS_Octet * value_out,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get the value of the given field, which is of type **DDS_Octet** (p. 1202).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>value_out</i>	<< <i>out</i> >> (p. 1013) If this method returned success, this argument will contain the value of the indicated member. It cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_octet (p. 399)

5.14.4.44 DDS_DynamicData_get_longlong()

```
DDS_ReturnCode_t DDS_DynamicData_get_longlong (
    const DDS_DynamicData * self,
    DDS_LongLong * value_out,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get the value of the given field, which is of type **DDS_LongLong** (p. 1203) or another type implicitly convertible to it (**DDS_Octet** (p. 1202), **DDS_Char** (p. 1201), **DDS_Short** (p. 1202), **DDS_UnsignedShort** (p. 1202), **DDS_Long** (p. 1203), **DDS_UnsignedLong** (p. 1203), or **DDS_Enum** (p. 1204)).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>value_out</i>	<< <i>out</i> >> (p. 1013) If this method returned success, this argument will contain the value of the indicated member. It cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_ulonglong (p. 400)

5.14.4.45 DDS_DynamicData_get_ulonglong()

```
DDS_ReturnCode_t DDS_DynamicData_get_ulonglong (
    const DDS_DynamicData * self,
    DDS_UnsignedLongLong * value_out,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get the value of the given field, which is of type **DDS_UnsignedLongLong** (p. 1203) or another type implicitly convertible to it (**DDS_Octet** (p. 1202), **DDS_Char** (p. 1201), **DDS_Short** (p. 1202), **DDS_UnsignedShort** (p. 1202), **DDS_Long** (p. 1203), **DDS_UnsignedLong** (p. 1203), or **DDS_Enum** (p. 1204)).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>value_out</i>	<< <i>out</i> >> (p. 1013) If this method returned success, this argument will contain the value of the indicated member. It cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i> Generated by Doxygen	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_ulonglong (p. 401)

5.14.4.46 **DDS_DynamicData_get_longdouble()**

```
DDS_ReturnCode_t DDS_DynamicData_get_longdouble (
    const DDS_DynamicData * self,
    DDS_LongDouble * value_out,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get the value of the given field, which is of type **DDS_LongDouble** (p. 1204) or another type implicitly convertible to it (**DDS_Float** (p. 1203) or **DDS_Double** (p. 1203)).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>value_out</i>	<< <i>out</i> >> (p. 1013) If this method returned success, this argument will contain the value of the indicated member. It cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_longdouble (p. 402)

5.14.4.47 DDS_DynamicData_get_wchar()

```
DDS_ReturnCode_t DDS_DynamicData_get_wchar (
    const DDS_DynamicData * self,
    DDS_Wchar * value_out,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get the value of the given field, which is of type **DDS_Wchar** (p. 1202) or another type implicitly convertible to it (**DDS_↵_Char** (p. 1201)).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>value_out</i>	<< <i>out</i> >> (p. 1013) If this method returned success, this argument will contain the value of the indicated member. It cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_wchar (p. 402)

5.14.4.48 DDS_DynamicData_get_int8()

```
DDS_ReturnCode_t DDS_DynamicData_get_int8 (
    const DDS_DynamicData * self,
    DDS_Int8 * value_out,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get the value of the given field, which is of type **DDS_Int8** (p. 1202).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>value_out</i>	<< <i>out</i> >> (p. 1013) If this method returned success, this argument will contain the value of the indicated member. It cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_int8 (p. 403)

5.14.4.49 DDS_DynamicData_get_uint8()

```
DDS_ReturnCode_t DDS_DynamicData_get_uint8 (
    const DDS_DynamicData * self,
    DDS_UInt8 * value_out,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get the value of the given field, which is of type **DDS_UInt8** (p. 1202).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>value_out</i>	<< <i>out</i> >> (p. 1013) If this method returned success, this argument will contain the value of the indicated member. It cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_uint8 (p. 404)

5.14.4.50 DDS_DynamicData_get_string()

```
DDS_ReturnCode_t DDS_DynamicData_get_string (
    const DDS_DynamicData * self,
    char ** value,
    DDS_UnsignedLong * size,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

Parameters

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>value</i>	<< <i>out</i> >> (p. 1013) The string into which the middleware should copy the string member. The allocated size of this string is indicated by <i>size</i> argument. If the size is sufficient to hold the contents of the member, they will be copied into this string. If <i>value</i> 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 <i>not</i> free your string; instead, this operation will fail and <i>size</i> will contain the minimum required size.
<i>size</i>	<< <i>inout</i> >> (p. 1013) As an input argument, the allocated size of the string. As an output argument, the actual size of the contents.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

See also

DDS_String_alloc (p. 1528)

DDS_String_free (p. 1529)

DDS_DynamicData_set_string (p. 405)

5.14.4.51 DDS_DynamicData_get_wstring()

```
DDS_ReturnCode_t DDS_DynamicData_get_wstring (
    const DDS_DynamicData * self,
    DDS_Wchar ** value,
    DDS_UnsignedLong * size,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get the value of the given field, which is of type **DDS_Wchar** (p. 1202)*.

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>value</i>	<< <i>out</i> >> (p. 1013) The string into which the middleware should copy the string member. The allocated size of this string is indicated by <i>size</i> argument. If the size is sufficient to hold the contents of the member, they will be copied into this string. If <i>value</i> 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 <i>not</i> free your string; instead, this operation will fail and <i>size</i> will contain the minimum required size.
<i>size</i>	<< <i>inout</i> >> (p. 1013) As an input argument, the allocated size of the string. As an output argument, the actual size of the contents.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

See also

DDS_String_alloc (p. 1528)

DDS_String_free (p. 1529)

DDS_DynamicData_set_wstring (p. 405)

5.14.4.52 DDS_DynamicData_get_complex_member()

```
DDS_ReturnCode_t DDS_DynamicData_get_complex_member (
    const DDS_DynamicData * self,
    DDS_DynamicData * value_out,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

The member may be of type kind **DDS_TK_ARRAY** (p. 243), **DDS_TK_SEQUENCE** (p. 243), **DDS_TK_STRUCT** (p. 243), **DDS_TK_VALUE** (p. 243), or **DDS_TK_UNION** (p. 243). It may be specified by name or by ID.

This method is logically related to **DDS_DynamicData_bind_complex_member** (p. 342) 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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>value_out</i>	<< <i>out</i> >> (p. 1013) The DDS_DynamicData (p. 1768) sample whose contents will be overwritten by this operation. This object must <i>not</i> be a bound member of another DDS_DynamicData (p. 1768) sample.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

See also

DDS_DynamicData_set_complex_member (p. 406)

DDS_DynamicData_bind_complex_member (p. 342)

5.14.4.53 DDS_DynamicData_get_long_array()

```
DDS_ReturnCode_t DDS_DynamicData_get_long_array (
    const DDS_DynamicData * self,
    DDS_Long * array,
    DDS_UnsignedLong * length,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get a copy of the given array member. The array may contain members of type **DDS_Long** (p. 1203) or **DDS_Enum** (p. 1204).

This method will perform an automatic conversion from **DDS_LongSeq** (p. 1835).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>array</i>	<< <i>out</i> >> (p. 1013) An already-allocated array, into which the elements will be copied.
<i>length</i>	<< <i>inout</i> >> (p. 1013) As an input, the allocated length of <i>array</i> . As an output, the number of elements that were copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_long_array (p. 408)

DDS_DynamicData_get_long_seq (p. 381)

5.14.4.54 DDS_DynamicData_get_short_array()

```
DDS_ReturnCode_t DDS_DynamicData_get_short_array (
    const DDS_DynamicData * self,
    DDS_Short * array,
    DDS_UnsignedLong * length,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_ShortSeq** (p. 2005).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>array</i>	<< <i>out</i> >> (p. 1013) An already-allocated array, into which the elements will be copied.
<i>length</i>	<< <i>inout</i> >> (p. 1013) As an input, the allocated length of <i>array</i> . As an output, the number of elements that were copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_short_array (p. 408)

DDS_DynamicData_get_short_seq (p. 382)

5.14.4.55 DDS_DynamicData_get_ulong_array()

```
DDS_ReturnCode_t DDS_DynamicData_get_ulong_array (
    const DDS_DynamicData * self,
    DDS_UnsignedLong * array,
    DDS_UnsignedLong * length,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get a copy of the given array member. The array may contain members of type **DDS_UnsignedLong** (p. 1203) or **DDS_Enum** (p. 1204).

This method will perform an automatic conversion from **DDS_UnsignedLongSeq** (p. 2086).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>array</i>	<< <i>out</i> >> (p. 1013) An already-allocated array, into which the elements will be copied.
<i>length</i>	<< <i>inout</i> >> (p. 1013) As an input, the allocated length of <i>array</i> . As an output, the number of elements that were copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_ulong_array (p. 409)

DDS_DynamicData_get_ulong_seq (p. 383)

5.14.4.56 DDS_DynamicData_get_ushort_array()

```
DDS_ReturnCode_t DDS_DynamicData_get_ushort_array (
    const DDS_DynamicData * self,
    DDS_UnsignedShort * array,
    DDS_UnsignedLong * length,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_UnsignedShortSeq** (p. 2086).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>array</i>	<< <i>out</i> >> (p. 1013) An already-allocated array, into which the elements will be copied.
<i>length</i>	<< <i>inout</i> >> (p. 1013) As an input, the allocated length of <i>array</i> . As an output, the number of elements that were copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_ushort_array (p. 410)

DDS_DynamicData_get_ushort_seq (p. 383)

5.14.4.57 DDS_DynamicData_get_float_array()

```
DDS_ReturnCode_t DDS_DynamicData_get_float_array (
    const DDS_DynamicData * self,
    DDS_Float * array,
    DDS_UnsignedLong * length,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_FloatSeq** (p. 1798).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>array</i>	<< <i>out</i> >> (p. 1013) An already-allocated array, into which the elements will be copied.
<i>length</i>	<< <i>inout</i> >> (p. 1013) As an input, the allocated length of <i>array</i> . As an output, the number of elements that were copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_float_array (p. 411)

DDS_DynamicData_get_float_seq (p. 384)

5.14.4.58 DDS_DynamicData_get_double_array()

```
DDS_ReturnCode_t DDS_DynamicData_get_double_array (
    const DDS_DynamicData * self,
    DDS_Double * array,
    DDS_UnsignedLong * length,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_DoubleSeq** (p. 1759).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>array</i>	<< <i>out</i> >> (p. 1013) An already-allocated array, into which the elements will be copied.
<i>length</i>	<< <i>inout</i> >> (p. 1013) As an input, the allocated length of <i>array</i> . As an output, the number of elements that were copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_double_array (p. 412)

DDS_DynamicData_get_double_seq (p. 385)

5.14.4.59 DDS_DynamicData_get_boolean_array()

```
DDS_ReturnCode_t DDS_DynamicData_get_boolean_array (
    const DDS_DynamicData * self,
    DDS_Boolean * array,
    DDS_UnsignedLong * length,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_BooleanSeq** (p. 1556).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>array</i>	<< <i>out</i> >> (p. 1013) An already-allocated array, into which the elements will be copied.
<i>length</i>	<< <i>inout</i> >> (p. 1013) As an input, the allocated length of <i>array</i> . As an output, the number of elements that were copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_boolean_array (p. 413)

DDS_DynamicData_get_boolean_seq (p. 386)

5.14.4.60 **DDS_DynamicData_get_char_array()**

```
DDS_ReturnCode_t DDS_DynamicData_get_char_array (
    const DDS_DynamicData * self,
    DDS_Char * array,
    DDS_UnsignedLong * length,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_CharSeq** (p. 1566).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>array</i>	<< <i>out</i> >> (p. 1013) An already-allocated array, into which the elements will be copied.
<i>length</i>	<< <i>inout</i> >> (p. 1013) As an input, the allocated length of <i>array</i> . As an output, the number of elements that were copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_char_array (p. 414)

DDS_DynamicData_get_char_seq (p. 387)

5.14.4.61 **DDS_DynamicData_get_octet_array()**

```
DDS_ReturnCode_t DDS_DynamicData_get_octet_array (
    const DDS_DynamicData * self,
    DDS_Octet * array,
    DDS_UnsignedLong * length,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_OctetSeq** (p. 1860).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>array</i>	<< <i>out</i> >> (p. 1013) An already-allocated array, into which the elements will be copied.
<i>length</i>	<< <i>inout</i> >> (p. 1013) As an input, the allocated length of <i>array</i> . As an output, the number of elements that were copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_octet_array (p. 415)

DDS_DynamicData_get_octet_seq (p. 388)

5.14.4.62 **DDS_DynamicData_get_longlong_array()**

```
DDS_ReturnCode_t DDS_DynamicData_get_longlong_array (
    const DDS_DynamicData * self,
    DDS_LongLong * array,
    DDS_UnsignedLong * length,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_LongLongSeq** (p. 1835).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>array</i>	<< <i>out</i> >> (p. 1013) An already-allocated array, into which the elements will be copied.
<i>length</i>	<< <i>inout</i> >> (p. 1013) As an input, the allocated length of <i>array</i> . As an output, the number of elements that were copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_longlong_array (p. 415)

DDS_DynamicData_get_longlong_seq (p. 389)

5.14.4.63 DDS_DynamicData_get_ulonglong_array()

```
DDS_ReturnCode_t DDS_DynamicData_get_ulonglong_array (
    const DDS_DynamicData * self,
    DDS_UnsignedLongLong * array,
    DDS_UnsignedLong * length,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_UnsignedLongLongSeq** (p. 2085).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>array</i>	<< <i>out</i> >> (p. 1013) An already-allocated array, into which the elements will be copied.
<i>length</i>	<< <i>inout</i> >> (p. 1013) As an input, the allocated length of <i>array</i> . As an output, the number of elements that were copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_ulonglong_array (p. 416)

DDS_DynamicData_get_ulonglong_seq (p. 389)

5.14.4.64 DDS_DynamicData_get_longdouble_array()

```
DDS_ReturnCode_t DDS_DynamicData_get_longdouble_array (
    const DDS_DynamicData * self,
    DDS_LongDouble * array,
    DDS_UnsignedLong * length,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_LongDoubleSeq** (p. 1835).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>array</i>	<< <i>out</i> >> (p. 1013) An already-allocated array, into which the elements will be copied.
<i>length</i>	<< <i>inout</i> >> (p. 1013) As an input, the allocated length of <i>array</i> . As an output, the number of elements that were copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_longdouble_array (p. 417)

DDS_DynamicData_get_longdouble_seq (p. 390)

5.14.4.65 **DDS_DynamicData_get_wchar_array()**

```
DDS_ReturnCode_t DDS_DynamicData_get_wchar_array (
    const DDS_DynamicData * self,
    DDS_Wchar * array,
    DDS_UnsignedLong * length,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_WcharSeq** (p. 2093).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>array</i>	<< <i>out</i> >> (p. 1013) An already-allocated array, into which the elements will be copied.
<i>length</i>	<< <i>inout</i> >> (p. 1013) As an input, the allocated length of <i>array</i> . As an output, the number of elements that were copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_wchar_array (p. 418)

DDS_DynamicData_get_wchar_seq (p. 391)

5.14.4.66 DDS_DynamicData_get_int8_array()

```
DDS_ReturnCode_t DDS_DynamicData_get_int8_array (
    const DDS_DynamicData * self,
    DDS_Int8 * array,
    DDS_UnsignedLong * length,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_Int8Seq** (p. 1810).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>array</i>	<< <i>out</i> >> (p. 1013) An already-allocated array, into which the elements will be copied.
<i>length</i>	<< <i>inout</i> >> (p. 1013) As an input, the allocated length of <i>array</i> . As an output, the number of elements that were copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_int8_array (p. 419)

DDS_DynamicData_get_int8_seq (p. 392)

5.14.4.67 DDS_DynamicData_get_uint8_array()

```
DDS_ReturnCode_t DDS_DynamicData_get_uint8_array (
    const DDS_DynamicData * self,
    DDS_UInt8 * array,
    DDS_UnsignedLong * length,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS_UInt8Seq** (p. 2083).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>array</i>	<< <i>out</i> >> (p. 1013) An already-allocated array, into which the elements will be copied.
<i>length</i>	<< <i>inout</i> >> (p. 1013) As an input, the allocated length of <i>array</i> . As an output, the number of elements that were copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222)

See also

DDS_DynamicData_set_uint8_array (p. 420)

DDS_DynamicData_get_uint8_seq (p. 393)

5.14.4.68 DDS_DynamicData_get_long_seq()

```
DDS_ReturnCode_t DDS_DynamicData_get_long_seq (
    const DDS_DynamicData * self,
    struct DDS_LongSeq * seq,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

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. 1203).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>seq</i>	<< <i>out</i> >> (p. 1013) A sequence, into which the elements will be copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

See also

DDS_DynamicData_set_long_seq (p. 421)

DDS_DynamicData_get_long_array (p. 366)

5.14.4.69 DDS_DynamicData_get_short_seq()

```
DDS_ReturnCode_t DDS_DynamicData_get_short_seq (
    const DDS_DynamicData * self,
    struct DDS_ShortSeq * seq,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

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. 1202).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>seq</i>	<< <i>out</i> >> (p. 1013) A sequence, into which the elements will be copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

See also

DDS_DynamicData_set_short_seq (p. 421)

DDS_DynamicData_get_short_array (p. 367)

5.14.4.70 DDS_DynamicData_get_ulong_seq()

```
DDS_ReturnCode_t DDS_DynamicData_get_ulong_seq (
    const DDS_DynamicData * self,
    struct DDS_UnsignedLongSeq * seq,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

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. 1203).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_←PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>seq</i>	<< <i>out</i> >> (p. 1013) A sequence, into which the elements will be copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

See also

DDS_DynamicData_set_ulong_seq (p. 422)

DDS_DynamicData_get_ulong_array (p. 368)

5.14.4.71 DDS_DynamicData_get_ushort_seq()

```
DDS_ReturnCode_t DDS_DynamicData_get_ushort_seq (
    const DDS_DynamicData * self,
    struct DDS_UnsignedShortSeq * seq,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

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. 1202).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↔PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>seq</i>	<< <i>out</i> >> (p. 1013) A sequence, into which the elements will be copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

See also

DDS_DynamicData_set_ushort_seq (p. 423)

DDS_DynamicData_get_ushort_array (p. 369)

5.14.4.72 DDS_DynamicData_get_float_seq()

```
DDS_ReturnCode_t DDS_DynamicData_get_float_seq (
    const DDS_DynamicData * self,
```



```
struct DDS_FloatSeq * seq,
const char * member_name,
DDS_DynamicDataMemberId member_id )
```

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_Float** (p. 1203).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>seq</i>	<< <i>out</i> >> (p. 1013) A sequence, into which the elements will be copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

See also

DDS_DynamicData_set_float_seq (p. 424)

DDS_DynamicData_get_float_array (p. 370)

5.14.4.73 DDS_DynamicData_get_double_seq()

```
DDS_ReturnCode_t DDS_DynamicData_get_double_seq (
const DDS_DynamicData * self,
struct DDS_DoubleSeq * seq,
const char * member_name,
DDS_DynamicDataMemberId member_id )
```

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. 1203).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>seq</i>	<< <i>out</i> >> (p. 1013) A sequence, into which the elements will be copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

See also

DDS_DynamicData_set_double_seq (p. 424)

DDS_DynamicData_get_double_array (p. 371)

5.14.4.74 DDS_DynamicData_get_boolean_seq()

```
DDS_ReturnCode_t DDS_DynamicData_get_boolean_seq (
    const DDS_DynamicData * self,
    struct DDS_BooleanSeq * seq,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

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. 1204).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>seq</i>	<< <i>out</i> >> (p. 1013) A sequence, into which the elements will be copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

See also

DDS_DynamicData_set_boolean_seq (p. 425)

DDS_DynamicData_get_boolean_array (p. 372)

5.14.4.75 DDS_DynamicData_get_char_seq()

```
DDS_ReturnCode_t DDS_DynamicData_get_char_seq (
    const   DDS_DynamicData * self,
    struct   DDS_CharSeq * seq,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

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. 1201).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>seq</i>	<< <i>out</i> >> (p. 1013) A sequence, into which the elements will be copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

See also

DDS_DynamicData_set_char_seq (p. 426)

DDS_DynamicData_get_char_array (p. 373)

5.14.4.76 **DDS_DynamicData_get_octet_seq()**

```
DDS_ReturnCode_t DDS_DynamicData_get_octet_seq (
    const DDS_DynamicData * self,
    struct DDS_OctetSeq * seq,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

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. 1202).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>seq</i>	<< <i>out</i> >> (p. 1013) A sequence, into which the elements will be copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

See also

DDS_DynamicData_set_octet_seq (p. 427)

DDS_DynamicData_get_octet_array (p. 374)

5.14.4.77 DDS_DynamicData_get_longlong_seq()

```
DDS_ReturnCode_t DDS_DynamicData_get_longlong_seq (
    const DDS_DynamicData * self,
    struct DDS_LongLongSeq * seq,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

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. 1203).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_←PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>seq</i>	<< <i>out</i> >> (p. 1013) A sequence, into which the elements will be copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

See also

DDS_DynamicData_set_longlong_seq (p. 427)

DDS_DynamicData_get_longlong_array (p. 375)

5.14.4.78 DDS_DynamicData_get_ulonglong_seq()

```
DDS_ReturnCode_t DDS_DynamicData_get_ulonglong_seq (
    const DDS_DynamicData * self,
    struct DDS_UnsignedLongLongSeq * seq,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

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. 1203).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↔PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>seq</i>	<< <i>out</i> >> (p. 1013) A sequence, into which the elements will be copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

See also

DDS_DynamicData_set_ulonglong_seq (p. 428)

DDS_DynamicData_get_ulonglong_array (p. 376)

5.14.4.79 DDS_DynamicData_get_longdouble_seq()

```
DDS_ReturnCode_t DDS_DynamicData_get_longdouble_seq (
    const DDS_DynamicData * self,
```

```
struct DDS_LongDoubleSeq * seq,
const char * member_name,
DDS_DynamicDataMemberId member_id )
```

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. 1204).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵ PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>seq</i>	<< <i>out</i> >> (p. 1013) A sequence, into which the elements will be copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

See also

DDS_DynamicData_set_longdouble_seq (p. 429)

DDS_DynamicData_get_longdouble_array (p. 377)

5.14.4.80 DDS_DynamicData_get_wchar_seq()

```
DDS_ReturnCode_t DDS_DynamicData_get_wchar_seq (
const DDS_DynamicData * self,
struct DDS_WcharSeq * seq,
const char * member_name,
DDS_DynamicDataMemberId member_id )
```

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. 1202).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵ PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>seq</i>	<< <i>out</i> >> (p. 1013) A sequence, into which the elements will be copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

See also

DDS_DynamicData_set_wchar_seq (p. 430)

DDS_DynamicData_get_wchar_array (p. 378)

5.14.4.81 DDS_DynamicData_get_int8_seq()

```
DDS_ReturnCode_t DDS_DynamicData_get_int8_seq (
    const DDS_DynamicData * self,
    struct DDS_Int8Seq * seq,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

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. 1202).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>seq</i>	<< <i>out</i> >> (p. 1013) A sequence, into which the elements will be copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

See also

DDS_DynamicData_set_int8_seq (p. 430)

DDS_DynamicData_get_int8_array (p. 379)

5.14.4.82 DDS_DynamicData_get_uint8_seq()

```
DDS_ReturnCode_t DDS_DynamicData_get_uint8_seq (
    const   DDS_DynamicData * self,
    struct   DDS_UInt8Seq * seq,
    const char * member_name,
    DDS_DynamicDataMemberId member_id )
```

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. 1202).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>seq</i>	<< <i>out</i> >> (p. 1013) A sequence, into which the elements will be copied.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).

Returns

One of the **Standard Return Codes** (p. 1221). If the member is optional and not set, this operation will return **DDS_RETCODE_NO_DATA** (p. 1222). This operation may also return **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

See also

DDS_DynamicData_set_uint8_seq (p. 431)

DDS_DynamicData_get_uint8_array (p. 380)

5.14.4.83 DDS_DynamicData_set_long()

```
DDS_ReturnCode_t DDS_DynamicData_set_long (
    DDS_DynamicData * self,
    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. 1203).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) The value to which to set the member.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_long (p. 351)

5.14.4.84 DDS_DynamicData_set_short()

```
DDS_ReturnCode_t DDS_DynamicData_set_short (
    DDS_DynamicData * self,
    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. 1202).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) The value to which to set the member.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_short (p. 351)

5.14.4.85 DDS_DynamicData_set_ulong()

```
DDS_ReturnCode_t DDS_DynamicData_set_ulong (
    DDS_DynamicData * self,
    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. 1203).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) The value to which to set the member.

See also

DDS_DynamicData_get_ulong (p. 352)

5.14.4.86 DDS_DynamicData_set_ushort()

```
DDS_ReturnCode_t DDS_DynamicData_set_ushort (
    DDS_DynamicData * self,
    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. 1202).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↔PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) The value to which to set the member.

See also

DDS_DynamicData_get_ushort (p. 353)

5.14.4.87 DDS_DynamicData_set_float()

```
DDS_ReturnCode_t DDS_DynamicData_set_float (
    DDS_DynamicData * self,
    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. 1203).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) The value to which to set the member.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_float (p. 354)

5.14.4.88 DDS_DynamicData_set_double()

```
DDS_ReturnCode_t DDS_DynamicData_set_double (
    DDS_DynamicData * self,
    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. 1203).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) The value to which to set the member.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_double (p. 355)

5.14.4.89 DDS_DynamicData_set_boolean()

```
DDS_ReturnCode_t DDS_DynamicData_set_boolean (
    DDS_DynamicData * self,
    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. 1204).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) The value to which to set the member.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_boolean (p. 356)

5.14.4.90 DDS_DynamicData_set_char()

```
DDS_ReturnCode_t DDS_DynamicData_set_char (
    DDS_DynamicData * self,
    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. 1201).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) The value to which to set the member.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_char (p. 356)

5.14.4.91 DDS_DynamicData_set_octet()

```
DDS_ReturnCode_t DDS_DynamicData_set_octet (
    DDS_DynamicData * self,
    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. 1202).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) The value to which to set the member.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_octet (p. 357)

5.14.4.92 DDS_DynamicData_set_longlong()

```
DDS_ReturnCode_t DDS_DynamicData_set_longlong (
    DDS_DynamicData * self,
    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. 1203).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) The value to which to set the member.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_ulonglong (p. 358)

5.14.4.93 DDS_DynamicData_set_ulonglong()

```
DDS_ReturnCode_t DDS_DynamicData_set_ulonglong (
    DDS_DynamicData * self,
    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. 1203).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) The value to which to set the member.

See also

DDS_DynamicData_get_ulonglong (p. 359)

5.14.4.94 DDS_DynamicData_set_longdouble()

```
DDS_ReturnCode_t DDS_DynamicData_set_longdouble (
    DDS_DynamicData * self,
    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. 1204).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) The value to which to set the member.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_longdouble (p. 360)

5.14.4.95 DDS_DynamicData_set_wchar()

```
DDS_ReturnCode_t DDS_DynamicData_set_wchar (
    DDS_DynamicData * self,
```

```
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. 1202).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) The value to which to set the member.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_char (p. 356)

5.14.4.96 DDS_DynamicData_set_int8()

```
DDS_ReturnCode_t DDS_DynamicData_set_int8 (
    DDS_DynamicData * self,
    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. 1202).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) The value to which to set the member.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_int8 (p. 361)

5.14.4.97 DDS_DynamicData_set_uint8()

```
DDS_ReturnCode_t DDS_DynamicData_set_uint8 (
    DDS_DynamicData * self,
    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. 1202).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) The value to which to set the member.

See also

DDS_DynamicData_get_uint8 (p. 362)

5.14.4.98 DDS_DynamicData_set_string()

```
DDS_ReturnCode_t DDS_DynamicData_set_string (
    DDS_DynamicData * self,
    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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) The value to which to set the member.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_string (p. 363)

5.14.4.99 DDS_DynamicData_set_wstring()

```
DDS_ReturnCode_t DDS_DynamicData_set_wstring (
    DDS_DynamicData * self,
```

```
const char * member_name,
    DDS_DynamicDataMemberId member_id,
const    DDS_Wchar * value )
```

Set the value of the given field of type **DDS_Wchar** (p. 1202)*.

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) The value to which to set the member.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_wstring (p. 364)

5.14.4.100 DDS_DynamicData_set_complex_member()

```
DDS_ReturnCode_t DDS_DynamicData_set_complex_member (
    DDS_DynamicData * self,
    const char * member_name,
    DDS_DynamicDataMemberId member_id,
    const    DDS_DynamicData * value )
```

Copy the state of the given **DDS_DynamicData** (p. 1768) 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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

The member may be of type kind **DDS_TK_ARRAY** (p. 243), **DDS_TK_SEQUENCE** (p. 243), **DDS_TK_STRUCT** (p. 243), **DDS_TK_VALUE** (p. 243), or **DDS_TK_UNION** (p. 243). It may be specified by name or by ID.

Example: Copying Data

This method can be used with **DDS_DynamicData_bind_complex_member** (p.342) to copy from one **DDS_DynamicData** (p.1768) 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.2109), `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).

```
struct DDS_DynamicData* foo_dst = ...;

struct DDS_DynamicData* foo_src = ...;

struct DDS_DynamicData* bar = DDS_DynamicData_new(NULL, myProperties);

/* Point to the source of the copy: */
DDS_DynamicData_bind_complex_member(
    foo_src,
    bar,
    "theBar",
    DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED);

/* Just one copy: */
DDS_DynamicData_set_complex_member(
    foo_dst,
    "theBar",
    DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED,
    bar);

/* Tear down: */
DDS_DynamicData_unbind_complex_member(foo_src, bar);

free(bar);
```

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p.1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p.1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p.1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p.324) to look up by name. See Member Names and IDs (p.1769).
<i>value</i>	<< <i>in</i> >> (p.1013) The source DDS_DynamicData (p.1768) object whose contents will be copied.

Returns

One of the **Standard Return Codes** (p.1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p.1222)

See also

DDS_DynamicData_get_complex_member (p. 365)

DDS_DynamicData_bind_complex_member (p. 342)

5.14.4.101 DDS_DynamicData_set_long_array()

```
DDS_ReturnCode_t DDS_DynamicData_set_long_array (
    DDS_DynamicData * self,
    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. 1203) or **DDS_Enum** (p. 1204).

This method will perform an automatic conversion to **DDS_LongSeq** (p. 1835).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>length</i>	<< <i>in</i> >> (p. 1013) The length of array.
<i>array</i>	<< <i>in</i> >> (p. 1013) The elements to copy.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_long_array (p. 366)

DDS_DynamicData_set_long_seq (p. 421)

5.14.4.102 DDS_DynamicData_set_short_array()

```
DDS_ReturnCode_t DDS_DynamicData_set_short_array (
    DDS_DynamicData * self,
    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. 2005).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>length</i>	<< <i>in</i> >> (p. 1013) The length of array.
<i>array</i>	<< <i>in</i> >> (p. 1013) The elements to copy.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_short_array (p. 367)

DDS_DynamicData_set_short_seq (p. 421)

5.14.4.103 DDS_DynamicData_set_ulong_array()

```
DDS_ReturnCode_t DDS_DynamicData_set_ulong_array (
    DDS_DynamicData * self,
```

```

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. 1203) or **DDS_Enum** (p. 1204).

This method will perform an automatic conversion to **DDS_UnsignedLongSeq** (p. 2086).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>length</i>	<< <i>in</i> >> (p. 1013) The length of array.
<i>array</i>	<< <i>in</i> >> (p. 1013) The elements to copy.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_ulong_array (p. 368)

DDS_DynamicData_set_ulong_seq (p. 422)

5.14.4.104 DDS_DynamicData_set_ushort_array()

```

DDS_ReturnCode_t DDS_DynamicData_set_ushort_array (
    DDS_DynamicData * self,
    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. 2086).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>length</i>	<< <i>in</i> >> (p. 1013) The length of <i>array</i> .
<i>array</i>	<< <i>in</i> >> (p. 1013) The elements to copy.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_ushort_array (p. 369)

DDS_DynamicData_set_ushort_seq (p. 423)

5.14.4.105 DDS_DynamicData_set_float_array()

```

DDS_ReturnCode_t DDS_DynamicData_set_float_array (
    DDS_DynamicData * self,
    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. 1798).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>length</i>	<< <i>in</i> >> (p. 1013) The length of <i>array</i> .
<i>array</i>	<< <i>in</i> >> (p. 1013) The elements to copy.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_float_array (p. 370)

DDS_DynamicData_set_float_seq (p. 424)

5.14.4.106 DDS_DynamicData_set_double_array()

```
DDS_ReturnCode_t DDS_DynamicData_set_double_array (
    DDS_DynamicData * self,
    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. 1759).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>length</i>	<< <i>in</i> >> (p. 1013) The length of <i>array</i> .
<i>array</i>	<< <i>in</i> >> (p. 1013) The elements to copy.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_double_array (p. 371)

DDS_DynamicData_set_double_seq (p. 424)

5.14.4.107 DDS_DynamicData_set_boolean_array()

```
DDS_ReturnCode_t DDS_DynamicData_set_boolean_array (
    DDS_DynamicData * self,
    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. 1556).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or
Generated by Doxygen	DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>length</i>	<< <i>in</i> >> (p. 1013) The length of <i>array</i> .
<i>array</i>	<< <i>in</i> >> (p. 1013) The elements to copy.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_boolean_array (p. 372)

DDS_DynamicData_set_boolean_seq (p. 425)

5.14.4.108 DDS_DynamicData_set_char_array()

```
DDS_ReturnCode_t DDS_DynamicData_set_char_array (
    DDS_DynamicData * self,
    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. 1566).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>length</i>	<< <i>in</i> >> (p. 1013) The length of array.
<i>array</i>	<< <i>in</i> >> (p. 1013) The elements to copy.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_char_array (p. 373)

DDS_DynamicData_set_char_seq (p. 426)

5.14.4.109 DDS_DynamicData_set_octet_array()

```
DDS_ReturnCode_t DDS_DynamicData_set_octet_array (
    DDS_DynamicData * self,
    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. 1860).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>length</i>	<< <i>in</i> >> (p. 1013) The length of array.
<i>array</i>	<< <i>in</i> >> (p. 1013) The elements to copy.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_octet_array (p. 374)

DDS_DynamicData_set_octet_seq (p. 427)

5.14.4.110 DDS_DynamicData_set_longlong_array()

```
DDS_ReturnCode_t DDS_DynamicData_set_longlong_array (
    DDS_DynamicData * self,
    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. 1835).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>length</i>	<< <i>in</i> >> (p. 1013) The length of array.
<i>array</i>	<< <i>in</i> >> (p. 1013) The elements to copy.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_longlong_array (p. 375)

DDS_DynamicData_set_longlong_seq (p. 427)

5.14.4.111 DDS_DynamicData_set_ulonglong_array()

```
DDS_ReturnCode_t DDS_DynamicData_set_ulonglong_array (
    DDS_DynamicData * self,
```



```
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. 2085).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>length</i>	<< <i>in</i> >> (p. 1013) The length of array.
<i>array</i>	<< <i>in</i> >> (p. 1013) The elements to copy.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_ulonglong_array (p. 376)

DDS_DynamicData_set_ulonglong_seq (p. 428)

5.14.4.112 DDS_DynamicData_set_longdouble_array()

```
DDS_ReturnCode_t DDS_DynamicData_set_longdouble_array (
    DDS_DynamicData * self,
    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. 1835).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>length</i>	<< <i>in</i> >> (p. 1013) The length of array.
<i>array</i>	<< <i>in</i> >> (p. 1013) The elements to copy.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_longdouble_array (p. 377)

DDS_DynamicData_set_longdouble_seq (p. 429)

5.14.4.113 DDS_DynamicData_set_wchar_array()

```
DDS_ReturnCode_t DDS_DynamicData_set_wchar_array (
    DDS_DynamicData * self,
    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. 2093).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>length</i>	<< <i>in</i> >> (p. 1013) The length of <i>array</i> .
<i>array</i>	<< <i>in</i> >> (p. 1013) The elements to copy.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_wchar_array (p. 378)

DDS_DynamicData_set_wchar_seq (p. 430)

5.14.4.114 DDS_DynamicData_set_int8_array()

```
DDS_ReturnCode_t DDS_DynamicData_set_int8_array (
    DDS_DynamicData * self,
    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. 1810).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>length</i>	<< <i>in</i> >> (p. 1013) The length of <i>array</i> .
<i>array</i>	<< <i>in</i> >> (p. 1013) The elements to copy.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_int8_array (p. 379)

DDS_DynamicData_set_int8_seq (p. 430)

5.14.4.115 **DDS_DynamicData_set_uint8_array()**

```
DDS_ReturnCode_t DDS_DynamicData_set_uint8_array (
    DDS_DynamicData * self,
    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. 2083).

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. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>length</i>	<< <i>in</i> >> (p. 1013) The length of <i>array</i> .
<i>array</i>	<< <i>in</i> >> (p. 1013) The elements to copy.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_uint8_array (p. 380)

DDS_DynamicData_set_uint8_seq (p. 431)

5.14.4.116 DDS_DynamicData_set_long_seq()

```
DDS_ReturnCode_t DDS_DynamicData_set_long_seq (
    DDS_DynamicData * self,
    const char * member_name,
    DDS_DynamicDataMemberId member_id,
    const struct 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. 1203).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) A sequence, from which the elements will be copied.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_long_seq (p. 381)

DDS_DynamicData_set_long_array (p. 408)

5.14.4.117 DDS_DynamicData_set_short_seq()

```
DDS_ReturnCode_t DDS_DynamicData_set_short_seq (
    DDS_DynamicData * self,
    const char * member_name,
    DDS_DynamicDataMemberId member_id,
    const struct 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. 1202).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) A sequence, from which the elements will be copied.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_short_seq (p. 382)

DDS_DynamicData_set_short_array (p. 408)

5.14.4.118 DDS_DynamicData_set_ulong_seq()

```
DDS_ReturnCode_t DDS_DynamicData_set_ulong_seq (
    DDS_DynamicData * self,
    const char * member_name,
    DDS_DynamicDataMemberId member_id,
    const struct 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. 1203).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) A sequence, from which the elements will be copied.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_ulong_seq (p. 383)

DDS_DynamicData_set_ulong_array (p. 409)

5.14.4.119 DDS_DynamicData_set_ushort_seq()

```
DDS_ReturnCode_t DDS_DynamicData_set_ushort_seq (
    DDS_DynamicData * self,
    const char * member_name,
    DDS_DynamicDataMemberId member_id,
    const struct 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. 1202).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or
Generated by Doxygen	DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) A sequence, from which the elements will be copied.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_ushort_seq (p. 383)

DDS_DynamicData_set_ushort_array (p. 410)

5.14.4.120 DDS_DynamicData_set_float_seq()

```
DDS_ReturnCode_t DDS_DynamicData_set_float_seq (
    DDS_DynamicData * self,
    const char * member_name,
    DDS_DynamicDataMemberId member_id,
    const struct 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. 1203).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) A sequence, from which the elements will be copied.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_float_seq (p. 384)

DDS_DynamicData_set_float_array (p. 411)

5.14.4.121 DDS_DynamicData_set_double_seq()

```
DDS_ReturnCode_t DDS_DynamicData_set_double_seq (
    DDS_DynamicData * self,
    const char * member_name,
    DDS_DynamicDataMemberId member_id,
    const struct 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. 1203).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) A sequence, from which the elements will be copied.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_double_seq (p. 385)

DDS_DynamicData_set_double_array (p. 412)

5.14.4.122 DDS_DynamicData_set_boolean_seq()

```
DDS_ReturnCode_t DDS_DynamicData_set_boolean_seq (
    DDS_DynamicData * self,
    const char * member_name,
    DDS_DynamicDataMemberId member_id,
    const struct 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. 1204).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) A sequence, from which the elements will be copied.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_boolean_seq (p. 386)

DDS_DynamicData_set_boolean_array (p. 413)

5.14.4.123 DDS_DynamicData_set_char_seq()

```
DDS_ReturnCode_t DDS_DynamicData_set_char_seq (
    DDS_DynamicData * self,
    const char * member_name,
    DDS_DynamicDataMemberId member_id,
    const struct 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. 1201).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) A sequence, from which the elements will be copied.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_char_seq (p. 387)

DDS_DynamicData_set_char_array (p. 414)

5.14.4.124 DDS_DynamicData_set_octet_seq()

```
DDS_ReturnCode_t DDS_DynamicData_set_octet_seq (
    DDS_DynamicData * self,
    const char * member_name,
    DDS_DynamicDataMemberId member_id,
    const struct 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. 1202).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) A sequence, from which the elements will be copied.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_octet_seq (p. 388)

DDS_DynamicData_set_octet_array (p. 415)

5.14.4.125 DDS_DynamicData_set_longlong_seq()

```
DDS_ReturnCode_t DDS_DynamicData_set_longlong_seq (
    DDS_DynamicData * self,
    const char * member_name,
    DDS_DynamicDataMemberId member_id,
    const struct 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. 1203).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) A sequence, from which the elements will be copied.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_longlong_seq (p. 389)

DDS_DynamicData_set_longlong_array (p. 415)

5.14.4.126 DDS_DynamicData_set_ulonglong_seq()

```
DDS_ReturnCode_t DDS_DynamicData_set_ulonglong_seq (
    DDS_DynamicData * self,
    const char * member_name,
    DDS_DynamicDataMemberId member_id,
    const struct 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. 1203).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) A sequence, from which the elements will be copied.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_ulonglong_seq (p. 389)

DDS_DynamicData_set_ulonglong_array (p. 416)

5.14.4.127 DDS_DynamicData_set_longdouble_seq()

```
DDS_ReturnCode_t DDS_DynamicData_set_longdouble_seq (
    DDS_DynamicData * self,
    const char * member_name,
    DDS_DynamicDataMemberId member_id,
    const struct 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. 1204).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or
Generated by Doxygen	DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) A sequence, from which the elements will be copied.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_longdouble_seq (p. 390)

DDS_DynamicData_set_longdouble_array (p. 417)

5.14.4.128 DDS_DynamicData_set_wchar_seq()

```
DDS_ReturnCode_t DDS_DynamicData_set_wchar_seq (
    DDS_DynamicData * self,
    const char * member_name,
    DDS_DynamicDataMemberId member_id,
    const struct 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. 1202).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) A sequence, from which the elements will be copied.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_wchar_seq (p. 391)

DDS_DynamicData_set_wchar_array (p. 418)

5.14.4.129 DDS_DynamicData_set_int8_seq()

```
DDS_ReturnCode_t DDS_DynamicData_set_int8_seq (
    DDS_DynamicData * self,
    const char * member_name,
    DDS_DynamicDataMemberId member_id,
    const struct 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. 1202).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) A sequence, from which the elements will be copied.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_int8_seq (p. 392)

DDS_DynamicData_set_int8_array (p. 419)

5.14.4.130 DDS_DynamicData_set_uint8_seq()

```
DDS_ReturnCode_t DDS_DynamicData_set_uint8_seq (
    DDS_DynamicData * self,
    const char * member_name,
    DDS_DynamicDataMemberId member_id,
    const struct 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. 1202).

This API is not supported when the DynamicData object is in CDR format and will fail with **DDS_RETCODE_↵PRECONDITION_NOT_MET** (p. 1222). See **DDS_DynamicData_is_cdr** (p. 333) for more information.

MT Safety:

UNSAFE.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>member_name</i>	<< <i>in</i> >> (p. 1013) The name of the member or NULL to look up the member by its ID.
<i>member_id</i>	<< <i>in</i> >> (p. 1013) The ID of the member or DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED (p. 324) to look up by name. See Member Names and IDs (p. 1769).
<i>value</i>	<< <i>in</i> >> (p. 1013) A sequence, from which the elements will be copied.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

See also

DDS_DynamicData_get_uint8_seq (p. 393)

DDS_DynamicData_set_uint8_array (p. 420)

5.14.4.131 DDS_DynamicDataTypeSupport_new()

```
struct DDS_DynamicDataTypeSupport * DDS_DynamicDataTypeSupport_new (
    const DDS_TypeCode * type,
    const struct DDS_DynamicDataTypeProperty_t * props )
```

Construct a new **DDS_DynamicDataTypeSupport** (p. 326) object.

This step is usually followed by type registration.

The **DDS_TypeCode** (p. 2071) 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. 2071) after this method returns.

Parameters

<i>type</i>	The DDS_TypeCode (p. 2071) that describes the members of this type. The new object will contain a <i>copy</i> of this DDS_TypeCode (p. 2071); you may delete the argument after this constructor returns.
<i>props</i>	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. 437).

See also

DDS_DynamicDataTypeSupport_register_type (p. 433)

DDS_DynamicDataTypeSupport_delete (p. 433)

Returns

One of the **Standard Return Codes** (p. 1221)

5.14.4.132 DDS_DynamicDataTypeSupport_delete()

```
void DDS_DynamicDataTypeSupport_delete (
    struct DDS_DynamicDataTypeSupport * self )
```

Delete a **DDS_DynamicDataTypeSupport** (p. 326) object.

A **DDS_DynamicDataTypeSupport** (p. 326) cannot be deleted while it is still in use. For each **DDS_DomainParticipant** (p. 77) with which the **DDS_DynamicDataTypeSupport** (p. 326) is registered, either the type must be unregistered or the participant must be deleted.

See also

DDS_DynamicDataTypeSupport_unregister_type (p. 433)

DDS_DynamicDataTypeSupport_new (p. 432)

5.14.4.133 DDS_DynamicDataTypeSupport_register_type()

```
DDS_ReturnCode_t DDS_DynamicDataTypeSupport_register_type (
    struct DDS_DynamicDataTypeSupport * self,
    DDS_DomainParticipant * participant,
    const char * type_name )
```

Associate the **DDS_TypeCode** (p. 2071) with the given **DDS_DomainParticipant** (p. 77) 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. 1222) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

See also

FooTypeSupport_register_type (p. 221)

DDS_DomainParticipant_create_topic (p. 115)

DDS_DynamicDataTypeSupport_unregister_type (p. 433)

5.14.4.134 DDS_DynamicDataTypeSupport_unregister_type()

```
DDS_ReturnCode_t DDS_DynamicDataTypeSupport_unregister_type (
    struct DDS_DynamicDataTypeSupport * self,
    DDS_DomainParticipant * participant,
    const char * type_name )
```

Remove the definition of this type from the **DDS_DomainParticipant** (p. 77).

This operation is optional; all types are automatically unregistered when a **DDS_DomainParticipant** (p. 77) 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 **DDS_Topic** (p. 175) is still referring to it.

If the type cannot be unregistered, this function will fail with **DDS_RETCODE_ERROR** (p. 1222) or **DDS_RETCODE_↵_BAD_PARAMETER** (p. 1222).

See also

FooTypeSupport_unregister_type (p. 222)

DDS_DynamicDataTypeSupport_register_type (p. 433)

5.14.4.135 DDS_DynamicDataTypeSupport_get_type_name()

```
const char * DDS_DynamicDataTypeSupport_get_type_name (
    const struct DDS_DynamicDataTypeSupport * self )
```

Get the default name of this type.

The **DDS_TypeCode** (p. 2071) that is wrapped by this **DDS_DynamicDataTypeSupport** (p. 326) 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.

```
DDS_DynamicDataTypeSupport_register_type(
    myTypeSupport,
    myParticipant,
    DDS_DynamicDataTypeSupport_get_type_name(myTypeSupport));
```

See also

FooTypeSupport_get_type_name (p. 221)

5.14.4.136 DDS_DynamicDataTypeSupport_get_data_type()

```
const struct DDS_TypeCode * DDS_DynamicDataTypeSupport_get_data_type (
    const struct DDS_DynamicDataTypeSupport * self )
```

Get the **DDS_TypeCode** (p. 2071) wrapped by this **DDS_DynamicDataTypeSupport** (p. 326).

5.14.4.137 DDS_DynamicDataTypeSupport_create_data()

```
DDS_DynamicData * DDS_DynamicDataTypeSupport_create_data (
    struct DDS_DynamicDataTypeSupport * self )
```

Create a new **DDS_DynamicData** (p. 1768) sample initialized with the **DDS_TypeCode** (p. 2071) and properties of this **DDS_DynamicDataTypeSupport** (p. 326).

This method returns a fully initialized object. It is *not* necessary to subsequently call **DDS_DynamicDataTypeSupport_initialize_data** (p. 436).

If initialization fails, this method returns NULL.

You must delete your **DDS_DynamicData** (p. 1768) object when you are finished with it.

```
struct DDS_DynamicData* sample = DDS_DynamicDataTypeSupport_create_data(
    myTypeSupport);
```

```
/* Failure indicated by NULL return result. */
```

```
/* Do something... */
```

```
DDS_DynamicDataTypeSupport_delete_data(myTypeSupport, sample);
```

See also

DDS_DynamicDataTypeSupport_initialize_data (p. 436)

DDS_DynamicDataTypeSupport_delete_data (p. 435)

FooTypeSupport_create_data (p. 215)

DDS_DynamicData_new (p. 327)

DDS_DynamicDataTypeProperty_t::data (p. 1781)

5.14.4.138 DDS_DynamicDataTypeSupport_delete_data()

```
DDS_ReturnCode_t DDS_DynamicDataTypeSupport_delete_data (
    struct DDS_DynamicDataTypeSupport * self,
    DDS_DynamicData * a_data )
```

Finalize and deallocate the **DDS_DynamicData** (p. 1768) sample.

This method fully finalizes the object. It is *not* necessary to previously call **DDS_DynamicDataTypeSupport_finalize_data** (p. 436).

See also

DDS_DynamicDataTypeSupport_finalize_data (p. 436)

FooTypeSupport_delete_data (p. 217)

DDS_DynamicDataTypeSupport_create_data (p. 435)

5.14.4.139 DDS_DynamicDataTypeSupport_print_data()

```
void DDS_DynamicDataTypeSupport_print_data (
    const struct DDS_DynamicDataTypeSupport * self,
    const DDS_DynamicData * a_data )
```

Print a string representation of the given sample to the given file.

This method is equivalent to **DDS_DynamicData_print** (p. 336).

See also

DDS_DynamicData_print (p. 336)

5.14.4.140 DDS_DynamicDataTypeSupport_copy_data()

```
DDS_ReturnCode_t DDS_DynamicDataTypeSupport_copy_data (
    const struct DDS_DynamicDataTypeSupport * self,
    DDS_DynamicData * dest,
    const DDS_DynamicData * source )
```

Deeply copy the given data samples.

5.14.4.141 DDS_DynamicDataTypeSupport_initialize_data()

```
DDS_ReturnCode_t DDS_DynamicDataTypeSupport_initialize_data (
    const struct DDS_DynamicDataTypeSupport * self,
    DDS_DynamicData * a_data )
```

Initialize a **DDS_DynamicData** (p. 1768) object to a valid empty state.

Use **DDS_DynamicDataTypeSupport_create_data** (p. 435) to allocate a new **DDS_DynamicData** (p. 1768) object on the heap. Use this method to initialize a **DDS_DynamicData** (p. 1768) object that is stored deeply, either within another object or on the stack.

When you're finished with the object, dispose of it with **DDS_DynamicDataTypeSupport_finalize_data** (p. 436).

```
struct DDS_DynamicData sample;

DDS_Boolean succeeded = DDS_DynamicDataTypeSupport_initialize_data(
    myTypeSupport, &sample);

/* Handle failure... */

/* Do something... */

DDS_DynamicDataTypeSupport_finalize_data(myTypeSupport, &sample);
```

See also

FooTypeSupport_initialize_data (p. 218)

DDS_DynamicDataTypeSupport_finalize_data (p. 436)

DDS_DynamicDataTypeSupport_create_data (p. 435)

5.14.4.142 DDS_DynamicDataTypeSupport_finalize_data()

```
DDS_ReturnCode_t DDS_DynamicDataTypeSupport_finalize_data (
    const struct DDS_DynamicDataTypeSupport * self,
    DDS_DynamicData * a_data )
```

Clean up the internal state of this object to get it ready for deallocation, but don't deallocate the pointer.

Use **DDS_DynamicDataTypeSupport_delete_data** (p. 435) to clean up a **DDS_DynamicData** (p. 1768) object on the heap. Use this method to clean up a **DDS_DynamicData** (p. 1768) object that is stored deeply, either within another object or on the stack.

See also

FooTypeSupport_finalize_data (p. 220)

DDS_DynamicDataTypeSupport_delete_data (p. 435)

DDS_DynamicDataTypeSupport_initialize_data (p. 436)

5.14.5 Variable Documentation

5.14.5.1 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. 1779).

Pass this object instead of your own **DDS_DynamicDataProperty_t** (p. 1779) object to use the default property values:

```
struct DDS_DynamicData* sample = DDS_DynamicData_new(
    myTypeCode,
    &DDS_DYNAMIC_DATA_PROPERTY_DEFAULT);
```

See also

DDS_DynamicDataProperty_t (p. 1779)

Examples

HelloWorldPlugin.c.

5.14.5.2 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. 1781).

Pass this object instead of your own **DDS_DynamicDataTypeProperty_t** (p. 1781) object to use the default property values:

```
struct DDS_DynamicDataTypeSupport* support = DDS_DynamicDataTypeSupport_new(
    myTypeCode,
    &DDS_DYNAMIC_DATA_TYPE_PROPERTY_DEFAULT);
```

See also

DDS_DynamicDataTypeProperty_t (p. 1781)

5.14.5.3 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. 1779).

Pass this object instead of your own **DDS_DynamicDataProperty_t** (p. 1779) object to use the default property values:

```
struct DDS_DynamicData* sample = DDS_DynamicData_new(
    myTypeCode,
    &DDS_DYNAMIC_DATA_PROPERTY_DEFAULT);
```

See also

DDS_DynamicDataProperty_t (p. 1779)

5.15 Publication Module

Contains the **DDS_FlowController** (p. 559), **DDS_Publisher** (p. 443), and **DDS_DataWriter** (p. 485) classes as well as the **DDS_PublisherListener** (p. 1917) and **DDS_DataWriterListener** (p. 1644) interfaces, and more generally, all that is needed on the publication side.

Modules

- **Publishers**
 - DDS_Publisher* (p. 443) entity and associated elements
- **Data Writers**
 - DDS_DataWriter* (p. 485) entity and associated elements
- **Flow Controllers**
 - <<extension>> (p. 1012) *DDS_FlowController* (p. 559) and associated elements
- **Multi-channel DataWriters**
 - APIs related to Multi-channel DataWriters.*

5.15.1 Detailed Description

Contains the **DDS_FlowController** (p. 559), **DDS_Publisher** (p. 443), and **DDS_DataWriter** (p. 485) classes as well as the **DDS_PublisherListener** (p. 1917) and **DDS_DataWriterListener** (p. 1644) interfaces, and more generally, all that is needed on the publication side.

"DCPS Publication package"

5.16 Publishers

DDS_Publisher (p. 443) entity and associated elements

Data Structures

- struct **DDS_PublisherSeq**
Declares IDL sequence < DDS_Publisher (p. 443) > .
- struct **DDS_PublisherQos**
QoS policies supported by a DDS_Publisher (p. 443) entity.
- struct **DDS_PublisherListener**
<<interface>> (p. 1013) DDS_Listener (p. 1816) for DDS_Publisher (p. 443) status.

Macros

- #define **DDS_PublisherQos_INITIALIZER**
Initializer for new QoS instances.
- #define **DDS_PublisherListener_INITIALIZER**
Initializer for new DDS_PublisherListener (p. 1917).

Typedefs

- typedef struct DDS_PublisherImpl **DDS_Publisher**
<<interface>> (p. 1013) 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. 1918) for equality.*
- **DDS_ReturnCode_t DDS_PublisherQos_print** (const struct **DDS_PublisherQos** *self)
*Prints this **DDS_PublisherQos** (p. 1918) to stdout.*
- **DDS_ReturnCode_t DDS_PublisherQos_to_string** (const struct **DDS_PublisherQos** *self, char *string, **DDS_UnsignedLong** *string_size)
*Obtains a string representation of this **DDS_PublisherQos** (p. 1918).*
- **DDS_ReturnCode_t DDS_PublisherQos_to_string_w_params** (const struct **DDS_PublisherQos** *self, char *string, **DDS_UnsignedLong** *string_size, const struct **DDS_PublisherQos** *base, const struct **DDS_QosPrintFormat** *format)
*Obtains a string representation of this **DDS_PublisherQos** (p. 1918).*
- **DDS_ReturnCode_t DDS_PublisherQos_initialize** (struct **DDS_PublisherQos** *self)
Initializer for new QoS instances.
- **DDS_ReturnCode_t DDS_PublisherQos_finalize** (struct **DDS_PublisherQos** *self)
*Free any dynamic memory allocated by the policies in this **DDS_PublisherQos** (p. 1918).*
- **DDS_ReturnCode_t DDS_PublisherQos_copy** (struct **DDS_PublisherQos** *self, const struct **DDS_PublisherQos** *source)
Copy the contents of the given QoS into this QoS.
- **DDS_Entity * DDS_Publisher_as_entity** (**DDS_Publisher** *publisher)
*Access a **DDS_Publisher** (p. 443)'s supertype instance.*
- **DDS_ReturnCode_t DDS_Publisher_get_default_datawriter_qos** (**DDS_Publisher** *self, struct **DDS_DataWriterQos** *qos)
*Copies the default **DDS_DataWriterQos** (p. 1666) values into the provided **DDS_DataWriterQos** (p. 1666) instance.*
- **DDS_ReturnCode_t DDS_Publisher_set_default_datawriter_qos** (**DDS_Publisher** *self, const struct **DDS_DataWriterQos** *qos)
*Sets the default **DDS_DataWriterQos** (p. 1666) values for this publisher.*
- **DDS_ReturnCode_t DDS_Publisher_set_default_datawriter_qos_with_profile** (**DDS_Publisher** *self, const char *library_name, const char *profile_name)
*<<extension>> (p. 1012) Set the default **DDS_DataWriterQos** (p. 1666) values for this publisher based on the input XML QoS profile.*
- **DDS_ReturnCode_t DDS_Publisher_set_default_profile** (**DDS_Publisher** *self, const char *library_name, const char *profile_name)
*<<extension>> (p. 1012) Sets the default XML profile for a **DDS_Publisher** (p. 443).*
- **DDS_ReturnCode_t DDS_Publisher_set_default_library** (**DDS_Publisher** *self, const char *library_name)
*<<extension>> (p. 1012) Sets the default XML library for a **DDS_Publisher** (p. 443).*
- **DDS_DataWriter * DDS_Publisher_create_datawriter** (**DDS_Publisher** *self, **DDS_Topic** *topic, const struct **DDS_DataWriterQos** *qos, const struct **DDS_DataWriterListener** *listener, **DDS_StatusMask** mask)
*Creates a **DDS_DataWriter** (p. 485) that will be attached and belong to the **DDS_Publisher** (p. 443).*
- **DDS_DataWriter * DDS_Publisher_create_datawriter_with_profile** (**DDS_Publisher** *self, **DDS_Topic** *topic, const char *library_name, const char *profile_name, const struct **DDS_DataWriterListener** *listener, **DDS_StatusMask** mask)
*<<extension>> (p. 1012) Creates a **DDS_DataWriter** (p. 485) object using the **DDS_DataWriterQos** (p. 1666) associated with the input XML QoS profile.*
- **DDS_ReturnCode_t DDS_Publisher_delete_datawriter** (**DDS_Publisher** *self, **DDS_DataWriter** *a_datawriter)
*Deletes a **DDS_DataWriter** (p. 485) that belongs to the **DDS_Publisher** (p. 443).*

- **DDS_DataWriter * DDS_Publisher_lookup_datawriter (DDS_Publisher *self, const char *topic_name)**
Retrieves the **DDS_DataWriter** (p. 485) for a specific **DDS_Topic** (p. 175).
- **DDS_ReturnCode_t DDS_Publisher_suspend_publications (DDS_Publisher *self)**
Indicates to RTI Connexx that the application is about to make multiple modifications using **DDS_DataWriter** (p. 485) objects belonging to the **DDS_Publisher** (p. 443).
- **DDS_ReturnCode_t DDS_Publisher_resume_publications (DDS_Publisher *self)**
Indicates to RTI Connexx that the application has completed the multiple changes initiated by the previous **DDS_Publisher_suspend_publications** (p. 457).
- **DDS_ReturnCode_t DDS_Publisher_begin_coherent_changes (DDS_Publisher *self)**
Indicates that the application will begin a coherent set of modifications using **DDS_DataWriter** (p. 485) objects attached to the **DDS_Publisher** (p. 443).
- **DDS_ReturnCode_t DDS_Publisher_end_coherent_changes (DDS_Publisher *self)**
Terminates the coherent set initiated by the matching call to **DDS_Publisher_begin_coherent_changes** (p. 459).
- **DDS_ReturnCode_t DDS_Publisher_get_all_datawriters (DDS_Publisher *self, struct DDS_DataWriterSeq *writers)**
Retrieve all the DataWriters created from this Publisher.
- **DDS_DomainParticipant * DDS_Publisher_get_participant (DDS_Publisher *self)**
Returns the **DDS_DomainParticipant** (p. 77) to which the **DDS_Publisher** (p. 443) belongs.
- **DDS_ReturnCode_t DDS_Publisher_delete_contained_entities (DDS_Publisher *self)**
Deletes all the entities that were created by means of the "create" operation on the **DDS_Publisher** (p. 443).
- **DDS_ReturnCode_t DDS_Publisher_copy_from_topic_qos (DDS_Publisher *self, struct DDS_DataWriterQos *a_datawriter_qos, const struct DDS_TopicQos *a_topic_qos)**
Copies the policies in the **DDS_TopicQos** (p. 2042) to the corresponding policies in the **DDS_DataWriterQos** (p. 1666).
- **DDS_ReturnCode_t DDS_Publisher_set_qos (DDS_Publisher *self, const struct DDS_PublisherQos *qos)**
Sets the publisher QoS.
- **DDS_ReturnCode_t DDS_Publisher_set_qos_with_profile (DDS_Publisher *self, const char *library_name, const char *profile_name)**
<<extension>> (p. 1012) Change the QoS of this publisher using the input XML QoS profile.
- **DDS_ReturnCode_t DDS_Publisher_get_qos (DDS_Publisher *self, struct DDS_PublisherQos *qos)**
Gets the publisher QoS.
- **const char * DDS_Publisher_get_default_library (DDS_Publisher *self)**
<<extension>> (p. 1012) Gets the default XML library associated with a **DDS_Publisher** (p. 443).
- **const char * DDS_Publisher_get_default_profile (DDS_Publisher *self)**
<<extension>> (p. 1012) Gets the default XML profile associated with a **DDS_Publisher** (p. 443).
- **const char * DDS_Publisher_get_default_profile_library (DDS_Publisher *self)**
<<extension>> (p. 1012) Gets the library where the default XML QoS profile is contained for a **DDS_Publisher** (p. 443).
- **DDS_ReturnCode_t DDS_Publisher_set_listener (DDS_Publisher *self, const struct DDS_PublisherListener *l, DDS_StatusMask mask)**
Sets the publisher listener.
- **struct DDS_PublisherListener DDS_Publisher_get_listener (DDS_Publisher *self)**
Get the publisher listener.
- **DDS_ReturnCode_t DDS_Publisher_get_listenerX (DDS_Publisher *self, struct DDS_PublisherListener *listener)**
<<extension>> (p. 1012) Get the publisher listener.
- **DDS_ReturnCode_t DDS_Publisher_wait_for_acknowledgments (DDS_Publisher *self, const struct DDS_Duration_t *max_wait)**
Blocks the calling thread until all data written by the Publisher's reliable DataWriters is acknowledged, or until timeout expires.

- **DDS_ReturnCode_t DDS_Publisher_wait_for_asynchronous_publishing** (**DDS_Publisher** *self, const struct **DDS_Duration_t** *max_wait)
<<extension>> (p. 1012) Blocks the calling thread until asynchronous sending is complete.
- **DDS_DataWriter * DDS_Publisher_lookup_datawriter_by_name** (**DDS_Publisher** *self, const char *datawriter_name)
*<<extension>> (p. 1012) Retrieves a **DDS_DataWriter** (p. 485) contained within the **DDS_Publisher** (p. 443) the **DDS_DataWriter** (p. 485) entity name.*

Variables

- const struct **DDS_DataWriterQos * DDS_DATAWRITER_QOS_PRINT_ALL**
*Special value which can be supplied to **DDS_DataWriterQos_to_string_w_params** (p. 534) indicating that all of the QoS should be printed.*
- const struct **DDS_DataWriterQos DDS_DATAWRITER_QOS_DEFAULT**
*Special value for creating **DDS_DataWriter** (p. 485) with default QoS.*
- const struct **DDS_DataWriterQos DDS_DATAWRITER_QOS_USE_TOPIC_QOS**
*Special value for creating **DDS_DataWriter** (p. 485) with a combination of the default **DDS_DataWriterQos** (p. 1666) and the **DDS_TopicQos** (p. 2042).*

5.16.1 Detailed Description

DDS_Publisher (p. 443) entity and associated elements

5.16.2 Macro Definition Documentation

5.16.2.1 DDS_PublisherQos_INITIALIZER

```
#define DDS_PublisherQos_INITIALIZER
```

Initializer for new QoS instances.

New **DDS_PublisherQos** (p. 1918) instance stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those contained QoS policies that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a new QoS structure is to initialize it on the stack at the time of its creation:

```
struct DDS_PublisherQos myQos = DDS_PublisherQos_INITIALIZER;
```

Note that the above assignment is not a substitute for calling **DDS_DomainParticipant_get_default_publisher_qos** (p. 89) or **DDS_Publisher_get_qos** (p. 464); one of those should be called subsequently to setting the QoS of a new or existing entity:

```
struct DDS_PublisherQos myQos = DDS_PublisherQos_INITIALIZER;
```

```
DDS_DomainParticipant_get_default_publisher_qos(myParticipant, &myQos);
```

```
DDS_Publisher_set_qos(myPub, &myQos);
```

```
DDS_PublisherQos_finalize(&myQos);
```

See also

DDS_DomainParticipant_get_default_publisher_qos (p. 89)

DDS_PublisherQos_finalize (p. 447)

5.16.2.2 DDS_PublisherListener_INITIALIZER

```
#define DDS_PublisherListener_INITIALIZER
```

Initializer for new **DDS_PublisherListener** (p. 1917).

All the new **DDS_PublisherListener** (p. 1917) instances allocated in the stack should be initialized to this value. No memory is allocated.

```
struct DDS_PublisherListener listener = DDS_PublisherListener_INITIALIZER;

/* initialize listener functions */

listener.as_datawriterlistener.on_offered_deadline_missed = ....;

DDS_Publisher_set_listener(myPublisher, &listener, mask);
```

See also

- DDS_Publisher_set_listener** (p. 466)
- DDS_PublisherListener** (p. 1917)
- Operations Allowed in Listener Callbacks** (p. 1818)

5.16.3 Typedef Documentation

5.16.3.1 DDS_Publisher

```
typedef struct DDS_PublisherImpl DDS_Publisher
```

<<*interface*>> (p. 1013) A publisher is the object responsible for the actual dissemination of publications.

QoS:

DDS_PublisherQos (p. 1918)

Listener:

DDS_PublisherListener (p. 1917)

A publisher acts on the behalf of one or several **DDS_DataWriter** (p. 485) objects that belong to it. When it is informed of a change to the data associated with one of its **DDS_DataWriter** (p. 485) 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 **DDS_Publisher** (p. 443) and the **DDS_DataWriter** (p. 485).

The following operations may be called even if the **DDS_Publisher** (p. 443) is not enabled. Other operations will fail with the value **DDS_RETCODE_NOT_ENABLED** (p. 1222) if called on a disabled **DDS_Publisher** (p. 443):

- **DDS_Entity_enable** (p. 1363),

- **DDS_Publisher_set_qos** (p. 462), **DDS_Publisher_get_qos** (p. 464) , **DDS_Publisher_set_qos_with_profile** (p. 463)
- **DDS_Publisher_set_listener** (p. 466), **DDS_Publisher_get_listener** (p. 466),
- **DDS_Entity_get_statuscondition** (p. 1365), **DDS_Entity_get_status_changes** (p. 1365)
- **DDS_Publisher_create_datawriter** (p. 453), **DDS_Publisher_create_datawriter_with_profile** (p. 455), **DDS_Publisher_delete_contained_entities** (p. 461), **DDS_Publisher_delete_datawriter** (p. 456),
- **DDS_Publisher_set_default_datawriter_qos** (p. 450), **DDS_Publisher_set_default_datawriter_qos_with_profile** (p. 450), **DDS_Publisher_get_default_datawriter_qos** (p. 449), **DDS_Publisher_wait_for_acknowledgments** (p. 467), **DDS_Publisher_set_default_library** (p. 452), **DDS_Publisher_set_default_profile** (p. 451),

See also

Operations Allowed in Listener Callbacks (p. 1818)

5.16.4 Function Documentation

5.16.4.1 DDS_PublisherQos_equals()

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

Compares two **DDS_PublisherQos** (p. 1918) for equality.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) This PublisherQos.
<i>other</i>	<< <i>in</i> >> (p. 1013) The other PublisherQos to be compared with this PublisherQos.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two Qos objects are equal or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.16.4.2 DDS_PublisherQos_print()

```
DDS_ReturnCode_t DDS_PublisherQos_print (
    const struct DDS_PublisherQos * self )
```

Prints this **DDS_PublisherQos** (p. 1918) to stdout.

Only the differences between this **DDS_PublisherQos** (p. 1918) and the documented default are printed. If you wish to print everything regardless, see **DDS_PublisherQos_to_string_w_params** (p. 446). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1926).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_RETCODE_OK (p. 1222) if no error was encountered.

5.16.4.3 DDS_PublisherQos_to_string()

```
DDS_ReturnCode_t DDS_PublisherQos_to_string (
    const struct DDS_PublisherQos * self,
    char * string,
    DDS_UnsignedLong * string_size )
```

Obtains a string representation of this **DDS_PublisherQos** (p. 1918).

Only the differences between this **DDS_PublisherQos** (p. 1918) and the documented default are printed to idref↩_PublisherQose string. If you wish to print everything regardless, see **DDS_PublisherQos_to_string_w_params** (p. 446). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1926).

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. 1918) is written to the buffer.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>string</i>	<< <i>out</i> >> (p. 1013) The char buffer that will be used to store the string representation of this DDS_PublisherQos (p. 1918). If NULL, this function will return the required length of this buffer through the string_size parameter.
<i>string_size</i>	<< <i>inout</i> >> (p. 1013) 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. 1222) if no error was encountered.

See also

DDS_PublisherQos_to_string_w_params (p. 446)

5.16.4.4 DDS_PublisherQos_to_string_w_params()

```
DDS_ReturnCode_t DDS_PublisherQos_to_string_w_params (
    const struct DDS_PublisherQos * self,
    char * string,
    DDS_UnsignedLong * string_size,
    const struct DDS_PublisherQos * base,
    const struct DDS_QosPrintFormat * format )
```

Obtains a string representation of this **DDS_PublisherQos** (p. 1918).

Only the differences between this **DDS_PublisherQos** (p. 1918) and the **DDS_PublisherQos** (p. 1918) supplied as the base are printed to the string.

It is possible to supply the sentinel value **DDS_PUBLISHER_QOS_PRINT_ALL** (p. 163) as the base to print everything within the QoS.

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. 1918) is written to the buffer.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>string</i>	<< <i>out</i> >> (p. 1013) The char buffer that will be used to store the string representation of this DDS_PublisherQos (p. 1918). If NULL, this function will return the required length of this buffer through the string_size parameter.
<i>string_size</i>	<< <i>inout</i> >> (p. 1013) 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.
<i>base</i>	<< <i>in</i> >> (p. 1013) The DDS_PublisherQos (p. 1918) to be used as the base QoS profile. Only the differences with respect to this base profile will be included in the output string.
<i>format</i>	<< <i>in</i> >> (p. 1013) The DDS_QosPrintFormat (p. 1926) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 1222) if no error was encountered.

5.16.4.5 DDS_PublisherQos_initialize()

```
DDS_ReturnCode_t DDS_PublisherQos_initialize (
    struct DDS_PublisherQos * self )
```

Initializer for new QoS instances.

New **DDS_PublisherQos** (p. 1918) instances on heap should be initialized with this function before they are passed to any function. This step ensures that those contained QoS policies that use dynamic memory are properly initialized. This function does not allocate memory.

Calling this function is not a substitute for calling **DDS_Publisher_get_qos** (p. 464) or **DDS_DomainParticipant_get_default_publisher_qos** (p. 89); one of those should be called subsequently to setting the QoS of any new or existing entity. **DDS_PublisherQos_finalize** (p. 447) should be called to free the contained QoS policies that use dynamic memory:

```
DDS_PublisherQos *myQos = malloc(sizeof(struct DDS_PublisherQos));

DDS_PublisherQos_initialize(myQos);

DDS_Publisher_get_default_datawriter_qos(myFactory, myQos);

DDS_Publisher_set_qos(myPublisher, myQos);

DDS_PublisherQos_finalize(myQos);

free(myQos);
```

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DomainParticipant_get_default_publisher_qos (p. 89)

DDS_PublisherQos_finalize (p. 447)

5.16.4.6 DDS_PublisherQos_finalize()

```
DDS_ReturnCode_t DDS_PublisherQos_finalize (
    struct DDS_PublisherQos * self )
```

Free any dynamic memory allocated by the policies in this **DDS_PublisherQos** (p. 1918).

Some QoS policies may use dynamic memory (regardless of whether the QoS itself is in dynamic memory). This function frees that memory but otherwise leaves this QoS unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

This function does not leave this object in an invalid state. It is permissible to clear a QoS and then subsequently allocate new dynamic memory in one or more of its QoS policies.

Note that if this QoS instance is stored in heap memory, calling this function will *not* call free() on it; the user is responsible for explicitly freeing any heap-based QoS instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_PublisherQos_INITIALIZER (p. 442)

DDS_PublisherQos_initialize (p. 446)

5.16.4.7 DDS_PublisherQos_copy()

```
DDS_ReturnCode_t DDS_PublisherQos_copy (
    struct DDS_PublisherQos * self,
    const struct DDS_PublisherQos * source )
```

Copy the contents of the given QoS into this QoS.

DDS_PublisherQos (p. 1918) instances can use dynamic memory because of the sequences contained in some QoS policies. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013) The DDS_PublisherQos (p. 1918) to copy from.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_PublisherQos_INITIALIZER (p. 442)

DDS_PublisherQos_initialize (p. 446)

DDS_PublisherQos_finalize (p. 447)

5.16.4.8 DDS_Publisher_as_entity()

```
DDS_Entity * DDS_Publisher_as_entity (
    DDS_Publisher * publisher )
```

Access a **DDS_Publisher** (p. 443)'s supertype instance.

Parameters

<i>publisher</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

The **DDS_Entity** (p. 1360) that is supertype instance of the publisher.

5.16.4.9 DDS_Publisher_get_default_datawriter_qos()

```
DDS_ReturnCode_t DDS_Publisher_get_default_datawriter_qos (
    DDS_Publisher * self,
    struct DDS_DataWriterQos * qos )
```

Copies the default **DDS_DataWriterQos** (p. 1666) values into the provided **DDS_DataWriterQos** (p. 1666) instance.

The retrieved *qos* will match the set of values specified on the last successful call to **DDS_Publisher_set_default_datawriter_qos** (p. 450) or **DDS_Publisher_set_default_datawriter_qos_with_profile** (p. 450), or else, if the call was never made, the default values from its owning **DDS_DomainParticipant** (p. 77).

This function 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 **DDS_Publisher** (p. 443) while another thread may be simultaneously calling **DDS_Publisher_set_default_datawriter_qos** (p. 450).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>inout</i> >> (p. 1013) DDS_DataWriterQos (p. 1666) to be filled-up. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DATAWRITER_QOS_DEFAULT (p. 470)

DDS_Publisher_create_datawriter (p. 453)

5.16.4.10 DDS_Publisher_set_default_datawriter_qos()

```
DDS_ReturnCode_t DDS_Publisher_set_default_datawriter_qos (
    DDS_Publisher * self,
    const struct DDS_DataWriterQos * qos )
```

Sets the default **DDS_DataWriterQos** (p. 1666) values for this publisher.

This call causes the default values inherited from the owning **DDS_DomainParticipant** (p. 77) to be overridden.

This default value will be used for newly created **DDS_DataWriter** (p. 485) if **DDS_DATAWRITER_QOS_DEFAULT** (p. 470) is specified as the `qos` parameter when **DDS_Publisher_create_datawriter** (p. 453) 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. 1222)

MT Safety:

UNSAFE. It is not safe to set the default QoS value from a **DDS_Publisher** (p. 443) while another thread may be simultaneously calling **DDS_Publisher_set_default_datawriter_qos** (p. 450), **DDS_Publisher_get_default_↵
datawriter_qos** (p. 449) or calling **DDS_Publisher_create_datawriter** (p. 453) with **DDS_DATAWRITER_QOS_↵
_DEFAULT** (p. 470) as the `qos` parameter.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>in</i> >> (p. 1013) Default qos to be set. The special value DDS_DATAREADER_QOS_DEFAULT (p. 600) may be passed as <code>qos</code> to indicate that the default QoS should be reset back to the initial values the factory would use if DDS_Publisher_set_default_datawriter_qos (p. 450) had never been called. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222)

5.16.4.11 DDS_Publisher_set_default_datawriter_qos_with_profile()

```
DDS_ReturnCode_t DDS_Publisher_set_default_datawriter_qos_with_profile (
    DDS_Publisher * self,
    const char * library_name,
    const char * profile_name )
```

<<**extension**>> (p. 1012) Set the default **DDS_DataWriterQos** (p. 1666) values for this publisher based on the input XML QoS profile.

This default value will be used for newly created **DDS_DataWriter** (p. 485) if **DDS_DATAWRITER_QOS_DEFAULT** (p. 470) is specified as the `qos` parameter when **DDS_Publisher_create_datawriter** (p. 453) is called.

Precondition

The **DDS_DataWriterQos** (p. 1666) 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. 1222)

MT Safety:

UNSAFE. It is not safe to set the default QoS value from a **DDS_Publisher** (p. 443) while another thread may be simultaneously calling **DDS_Publisher_set_default_datawriter_qos** (p. 450), **DDS_Publisher_get_default_datawriter_qos** (p. 449) or calling **DDS_Publisher_create_datawriter** (p. 453) with **DDS_DATAWRITER_QOS_DEFAULT** (p. 470) as the `qos` parameter.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) Library name containing the XML QoS profile. If <code>library_name</code> is null RTI Connexx will use the default library (see DDS_Publisher_set_default_library (p. 452)).
<i>profile_name</i>	<< in >> (p. 1013) XML QoS Profile name. If <code>profile_name</code> is null RTI Connexx will use the default profile (see DDS_Publisher_set_default_profile (p. 451)).

If the input profile cannot be found, the function fails with **DDS_RETCODE_ERROR** (p. 1222).

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222)

See also

DDS_DATAWRITER_QOS_DEFAULT (p. 470)

DDS_Publisher_create_datawriter_with_profile (p. 455)

5.16.4.12 DDS_Publisher_set_default_profile()

```
DDS_ReturnCode_t DDS_Publisher_set_default_profile (
    DDS_Publisher * self,
    const char * library_name,
    const char * profile_name )
```

<<**extension**>> (p. 1012) Sets the default XML profile for a **DDS_Publisher** (p. 443).

This function 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 **DDS_Publisher** (p. 443) function 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 **DDS_Publisher** (p. 443) inherits the default from the **DDS_DomainParticipant** (p. 77) (see **DDS_DomainParticipant_set_default_profile** (p. 102)).

This function does not set the default QoS for **DDS_DataWriter** (p. 485) objects created by the **DDS_Publisher** (p. 443); for this functionality, use **DDS_Publisher_set_default_datawriter_qos_with_profile** (p. 450) (you may pass in NULL after having called `set_default_profile()`).

This function does not set the default QoS for newly created Publishers; for this functionality, use **DDS_DomainParticipant_set_default_publisher_qos_with_profile** (p. 90).

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) The library name containing the profile.
<i>profile_name</i>	<< in >> (p. 1013) The profile name. If <code>profile_name</code> is null any previous default is unset.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_Publisher_get_default_profile (p. 465)

DDS_Publisher_get_default_profile_library (p. 465)

5.16.4.13 DDS_Publisher_set_default_library()

```
DDS_ReturnCode_t DDS_Publisher_set_default_library (
    DDS_Publisher * self,
    const char * library_name )
```

<<**extension**>> (p. 1012) Sets the default XML library for a **DDS_Publisher** (p. 443).

This function 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 **DDS_Publisher** (p. 443) inherits the default from the **DDS_DomainParticipant** (p. 77) (see **DDS_DomainParticipant_set_default_library** (p. 101)).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< <i>in</i> >> (p. 1013) Library name. If <code>library_name</code> is null any previous default is unset.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_Publisher_get_default_library (p. 464)

5.16.4.14 DDS_Publisher_create_datawriter()

```
DDS_DataWriter * DDS_Publisher_create_datawriter (
    DDS_Publisher * self,
    DDS_Topic * topic,
    const struct DDS_DataWriterQos * qos,
    const struct DDS_DataWriterListener * listener,
    DDS_StatusMask mask )
```

Creates a **DDS_DataWriter** (p. 485) that will be attached and belong to the **DDS_Publisher** (p. 443).

For each application-defined type, **Foo** (p. 2109), there is an implied, auto-generated class **FooDataWriter** (p. 2113) that extends **DDS_DataWriter** (p. 485) and contains the operations to write data of type **Foo** (p. 2109).

Note that a common application pattern to construct the QoS for the **DDS_DataWriter** (p. 485) is to:

- Retrieve the QoS policies on the associated **DDS_Topic** (p. 175) by means of the **DDS_Topic_get_qos** (p. 199) operation.
- Retrieve the default **DDS_DataWriter** (p. 485) qos by means of the **DDS_Publisher_get_default_datawriter_qos** (p. 449) operation.
- Combine those two QoS policies (for example, using **DDS_Publisher_copy_from_topic_qos** (p. 462)) and selectively modify policies as desired.

When a **DDS_DataWriter** (p. 485) is created, only those transports already registered are available to the **DDS_DataWriter** (p. 485). See **Built-in Transport Plugins** (p. 736) 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 **DDS_DataWriter** (p. 485) will be created.

The given **DDS_Topic** (p. 175) must have been created from the same participant as this publisher. If it was created from a different participant, this function will fail.

MT Safety:

UNSAFE. If **DDS_DATAWRITER_QOS_DEFAULT** (p. 470) is used for the `qos` parameter, it is not safe to create the datawriter while another thread may be simultaneously calling **DDS_Publisher_set_default_datawriter_qos** (p. 450).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>topic</i>	<< <i>in</i> >> (p. 1013) The DDS_Topic (p. 175) that the DDS_DataWriter (p. 485) will be associated with. Cannot be NULL.
<i>qos</i>	<< <i>in</i> >> (p. 1013) QoS to be used for creating the new DDS_DataWriter (p. 485). The special value DDS_DATAWRITER_QOS_DEFAULT (p. 470) can be used to indicate that the DDS_DataWriter (p. 485) should be created with the default DDS_DataWriterQos (p. 1666) set in the DDS_Publisher (p. 443). The special value DDS_DATAWRITER_QOS_USE_TOPIC_QOS (p. 470) can be used to indicate that the DDS_DataWriter (p. 485) should be created with the combination of the default DDS_DataWriterQos (p. 1666) set on the DDS_Publisher (p. 443) and the DDS_TopicQos (p. 2042) of the DDS_Topic (p. 175). Cannot be NULL.
<i>listener</i>	<< <i>in</i> >> (p. 1013) The listener of the DDS_DataWriter (p. 485).
<i>mask</i>	<< <i>in</i> >> (p. 1013). Changes of communication status to be invoked on the listener. See DDS_StatusMask (p. 1226).

Returns

A **DDS_DataWriter** (p. 485) of a derived class specific to the data type associated with the **DDS_Topic** (p. 175) or NULL if an error occurred.

See also

FooDataWriter (p. 2113)

Specifying QoS on entities (p. 1244) for information on setting QoS before entity creation

DDS_DataWriterQos (p. 1666) for rules on consistency among QoS

DDS_DATAWRITER_QOS_DEFAULT (p. 470)

DDS_DATAWRITER_QOS_USE_TOPIC_QOS (p. 470)

DDS_Publisher_create_datawriter_with_profile (p. 455)

DDS_Publisher_get_default_datawriter_qos (p. 449)

DDS_Topic_set_qos (p. 197)

DDS_Publisher_copy_from_topic_qos (p. 462)

DDS_DataWriter_set_listener (p. 553)

Examples

HelloWorld_publisher.c.

5.16.4.15 DDS_Publisher_create_datawriter_with_profile()

```
DDS_DataWriter * DDS_Publisher_create_datawriter_with_profile (
    DDS_Publisher * self,
    DDS_Topic * topic,
    const char * library_name,
    const char * profile_name,
    const struct DDS_DataWriterListener * listener,
    DDS_StatusMask mask )
```

<<**extension**>> (p. 1012) Creates a **DDS_DataWriter** (p. 485) object using the **DDS_DataWriterQos** (p. 1666) associated with the input XML QoS profile.

The **DDS_DataWriter** (p. 485) will be attached and belong to the **DDS_Publisher** (p. 443).

For each application-defined type, **Foo** (p. 2109), there is an implied, auto-generated class **FooDataWriter** (p. 2113) that extends **DDS_DataWriter** (p. 485) and contains the operations to write data of type **Foo** (p. 2109).

When a **DDS_DataWriter** (p. 485) is created, only those transports already registered are available to the **DDS_DataWriter** (p. 485). See **Built-in Transport Plugins** (p. 736) 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 **DDS_DataWriter** (p. 485) will be created.

The given **DDS_Topic** (p. 175) must have been created from the same participant as this publisher. If it was created from a different participant, this function will return NULL.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>topic</i>	<< in >> (p. 1013) The DDS_Topic (p. 175) that the DDS_DataWriter (p. 485) will be associated with. Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) Library name containing the XML QoS profile. If library_name is null RTI Connex will use the default library (see DDS_Publisher_set_default_library (p. 452)).
<i>profile_name</i>	<< in >> (p. 1013) XML QoS Profile name. If profile_name is null RTI Connex will use the default profile (see DDS_Publisher_set_default_profile (p. 451)).
<i>listener</i>	<< in >> (p. 1013) The listener of the DDS_DataWriter (p. 485).
<i>mask</i>	<< in >> (p. 1013). Changes of communication status to be invoked on the listener. See DDS_StatusMask (p. 1226).

Returns

A **DDS_DataWriter** (p. 485) of a derived class specific to the data type associated with the **DDS_Topic** (p. 175) or NULL if an error occurred.

See also

FooDataWriter (p. 2113)

Specifying QoS on entities (p. 1244) for information on setting QoS before entity creation

DDS_DataWriterQos (p. 1666) for rules on consistency among QoS

DDS_Publisher_create_datawriter (p. 453)

DDS_Publisher_get_default_datawriter_qos (p. 449)

DDS_Topic_set_qos (p. 197)

DDS_Publisher_copy_from_topic_qos (p. 462)

DDS_DataWriter_set_listener (p. 553)

5.16.4.16 DDS_Publisher_delete_datawriter()

```
DDS_ReturnCode_t DDS_Publisher_delete_datawriter (
    DDS_Publisher * self,
    DDS_DataWriter * a_datawriter )
```

Deletes a **DDS_DataWriter** (p. 485) that belongs to the **DDS_Publisher** (p. 443).

The deletion of the **DDS_DataWriter** (p. 485) will automatically unregister all instances.

Precondition

If the **DDS_DataWriter** (p. 485) does not belong to the **DDS_Publisher** (p. 443), the operation will fail with **DDS_↵_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

Postcondition

Listener installed on the **DDS_DataWriter** (p. 485) will not be called after this function 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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>a_datawriter</i>	<< <i>in</i> >> (p. 1013) The DDS_DataWriter (p. 485) to be deleted.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

5.16.4.17 DDS_Publisher_lookup_datawriter()

```
DDS_DataWriter * DDS_Publisher_lookup_datawriter (
    DDS_Publisher * self,
    const char * topic_name )
```

Retrieves the **DDS_DataWriter** (p. 485) for a specific **DDS_Topic** (p. 175).

This returned **DDS_DataWriter** (p. 485) is either enabled or disabled.

If more than one **DDS_DataWriter** (p. 485) is attached to the **DDS_Publisher** (p. 443) with the same `topic_name`, then this operation may return any one of them.

MT Safety:

UNSAFE. It is not safe to lookup a **DDS_DataWriter** (p. 485) in one thread while another thread is simultaneously creating or destroying that **DDS_DataWriter** (p. 485).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>topic_name</i>	<< <i>in</i> >> (p. 1013) Name of the DDS_Topic (p. 175) associated with the DDS_DataWriter (p. 485) that is to be looked up. Cannot be NULL.

Returns

A **DDS_DataWriter** (p. 485) that belongs to the **DDS_Publisher** (p. 443) attached to the **DDS_Topic** (p. 175) with `topic_name`. If no such **DDS_DataWriter** (p. 485) exists, this operation returns NULL.

5.16.4.18 DDS_Publisher_suspend_publications()

```
DDS_ReturnCode_t DDS_Publisher_suspend_publications (
    DDS_Publisher * self )
```

Indicates to RTI Connext that the application is about to make multiple modifications using **DDS_DataWriter** (p. 485) objects belonging to the **DDS_Publisher** (p. 443).

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 **DDS_Publisher_resume_publications** (p. 458) indicating that the set of modifications has completed.

If the **DDS_Publisher** (p. 443) is deleted before **DDS_Publisher_resume_publications** (p. 458) 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 **DDS_FlowController** (p. 559), **DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 1319) **DDS_DataWriter** (p. 485) instances allow the user even finer control of traffic shaping and sample coalescing.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

DDS_FlowController (p. 559)
DDS_FlowController_trigger_flow (p. 563)
DDS_ON_DEMAND_FLOW_CONTROLLER_NAME (p. 565)
DDS_PublishModeQosPolicy (p. 1920)

5.16.4.19 DDS_Publisher_resume_publications()

```
DDS_ReturnCode_t DDS_Publisher_resume_publications (
    DDS_Publisher * self )
```

Indicates to RTI Connext that the application has completed the multiple changes initiated by the previous **DDS_Publisher_suspend_publications** (p. 457).

This is a **hint** to RTI Connext that can be used for example, to batch all the modifications made since the **DDS_Publisher_suspend_publications** (p. 457).

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 **DDS_FlowController** (p. 559), **DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 1319) **DDS_DataWriter** (p. 485) instances allow the user even finer control of traffic shaping and sample coalescing.

Precondition

A call to **DDS_Publisher_resume_publications** (p. 458) must match a previous call to **DDS_Publisher_suspend_publications** (p. 457). Otherwise the operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

DDS_FlowController (p. 559)
DDS_FlowController_trigger_flow (p. 563)
DDS_ON_DEMAND_FLOW_CONTROLLER_NAME (p. 565)
DDS_PublishModeQosPolicy (p. 1920)

5.16.4.20 DDS_Publisher_begin_coherent_changes()

```
DDS_ReturnCode_t DDS_Publisher_begin_coherent_changes (
    DDS_Publisher * self )
```

Indicates that the application will begin a coherent set of modifications using **DDS_DataWriter** (p. 485) objects attached to the **DDS_Publisher** (p. 443).

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 **DDS_Publisher** (p. 443) or one of its subscribers (**DDS_Subscriber** (p. 572)) may change, a late-joining **DDS_↔DataReader** (p. 615) 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 **DDS_Publisher_end_↔_coherent_changes** (p. 460). 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 Data_↔Readers 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 **DDS_TopicQuery** (p. 707) 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.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

DDS_PresentationQosPolicy (p. 1890)

5.16.4.21 DDS_Publisher_end_coherent_changes()

```
DDS_ReturnCode_t DDS_Publisher_end_coherent_changes (
    DDS_Publisher * self )
```

Terminates the coherent set initiated by the matching call to **DDS_Publisher_begin_coherent_changes** (p. 459).

Precondition

If there is no matching call to **DDS_Publisher_begin_coherent_changes** (p. 459) the operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

5.16.4.22 DDS_Publisher_get_all_datawriters()

```
DDS_ReturnCode_t DDS_Publisher_get_all_datawriters (
    DDS_Publisher * self,
    struct DDS_DataWriterSeq * writers )
```

Retrieve all the DataWriters created from this Publisher.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>writers</i>	<< <i>inout</i> >> (p. 1013) Sequence where the DataWriters will be added

Returns

One of the **Standard Return Codes** (p. 1221)

5.16.4.23 DDS_Publisher_get_participant()

```
DDS_DomainParticipant * DDS_Publisher_get_participant (
    DDS_Publisher * self )
```

Returns the **DDS_DomainParticipant** (p. 77) to which the **DDS_Publisher** (p. 443) belongs.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

the **DDS_DomainParticipant** (p. 77) to which the **DDS_Publisher** (p. 443) belongs.

5.16.4.24 DDS_Publisher_delete_contained_entities()

```
DDS_ReturnCode_t DDS_Publisher_delete_contained_entities (
    DDS_Publisher * self )
```

Deletes all the entities that were created by means of the "create" operation on the **DDS_Publisher** (p. 443).

Deletes all contained **DDS_DataWriter** (p. 485) objects. Once **DDS_Publisher_delete_contained_entities** (p. 461) completes successfully, the application may delete the **DDS_Publisher** (p. 443), knowing that it has no contained **DDS_↵_DataWriter** (p. 485) objects.

The operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) 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.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

5.16.4.25 DDS_Publisher_copy_from_topic_qos()

```
DDS_ReturnCode_t DDS_Publisher_copy_from_topic_qos (
    DDS_Publisher * self,
    struct DDS_DataWriterQos * a_datawriter_qos,
    const struct DDS_TopicQos * a_topic_qos )
```

Copies the policies in the **DDS_TopicQos** (p. 2042) to the corresponding policies in the **DDS_DataWriterQos** (p. 1666).

Copies the policies in the **DDS_TopicQos** (p. 2042) to the corresponding policies in the **DDS_DataWriterQos** (p. 1666) (replacing values in the **DDS_DataWriterQos** (p. 1666), if present).

This is a "convenience" operation most useful in combination with the operations **DDS_Publisher_get_default_datawriter_qos** (p. 449) and **DDS_Topic_get_qos** (p. 199). The operation **DDS_Publisher_copy_from_topic_qos** (p. 462) can be used to merge the **DDS_DataWriter** (p. 485) default QoS policies with the corresponding ones on the **DDS_Topic** (p. 175). The resulting QoS can then be used to create a new **DDS_DataWriter** (p. 485), or set its QoS.

This operation does not check the resulting **DDS_DataWriterQos** (p. 1666) for consistency. This is because the 'merged' **DDS_DataWriterQos** (p. 1666) may not be the final one, as the application can still modify some policies prior to applying the policies to the **DDS_DataWriter** (p. 485).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>a_datawriter_qos</i>	<< <i>inout</i> >> (p. 1013) DDS_DataWriterQos (p. 1666) to be filled-up. Cannot be NULL.
<i>a_topic_qos</i>	<< <i>in</i> >> (p. 1013) DDS_TopicQos (p. 2042) to be merged with DDS_DataWriterQos (p. 1666). Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.16.4.26 DDS_Publisher_set_qos()

```
DDS_ReturnCode_t DDS_Publisher_set_qos (
    DDS_Publisher * self,
    const struct DDS_PublisherQos * qos )
```

Sets the publisher QoS.

This operation modifies the QoS of the **DDS_Publisher** (p. 443).

The **DDS_PublisherQos::group_data** (p. 1919), **DDS_PublisherQos::partition** (p. 1919) and **DDS_PublisherQos::entity_factory** (p. 1919) can be changed. The other policies are immutable.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>in</i> >> (p. 1013) DDS_PublisherQos (p. 1918) to be set to. Policies must be consistent. Immutable policies cannot be changed after DDS_Publisher (p. 443) is enabled. The special value DDS_PUBLISHER_QOS_DEFAULT (p. 161) can be used to indicate that the QoS of the DDS_Publisher (p. 443) should be changed to match the current default DDS_PublisherQos (p. 1918) set in the DDS_DomainParticipant (p. 77). Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 1222), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222).

See also

DDS_PublisherQos (p. 1918) for rules on consistency among QoS
set_qos (abstract) (p. 1361)
Operations Allowed in Listener Callbacks (p. 1818)

5.16.4.27 DDS_Publisher_set_qos_with_profile()

```
DDS_ReturnCode_t DDS_Publisher_set_qos_with_profile (
    DDS_Publisher * self,
    const char * library_name,
    const char * profile_name )
```

<<*extension*>> (p. 1012) Change the QoS of this publisher using the input XML QoS profile.

This operation modifies the QoS of the **DDS_Publisher** (p. 443).

The **DDS_PublisherQos::group_data** (p. 1919), **DDS_PublisherQos::partition** (p. 1919) and **DDS_PublisherQos::entity_factory** (p. 1919) can be changed. The other policies are immutable.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< <i>in</i> >> (p. 1013) Library name containing the XML QoS profile. If <i>library_name</i> is null RTI Connexx will use the default library (see DDS_Publisher_set_default_library (p. 452)).
<i>profile_name</i>	<< <i>in</i> >> (p. 1013) XML QoS Profile name. If <i>profile_name</i> is null RTI Connexx will use the default profile (see DDS_Publisher_set_default_profile (p. 451)).

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 1222), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222).

See also

DDS_PublisherQos (p. 1918) for rules on consistency among QoS
Operations Allowed in Listener Callbacks (p. 1818)

5.16.4.28 DDS_Publisher_get_qos()

```
DDS_ReturnCode_t DDS_Publisher_get_qos (
    DDS_Publisher * self,
    struct DDS_PublisherQos * qos )
```

Gets the publisher QoS.

This function may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>in</i> >> (p. 1013) DDS_PublisherQos (p. 1918) to be filled in. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

get_qos (abstract) (p. 1362)

5.16.4.29 DDS_Publisher_get_default_library()

```
const char * DDS_Publisher_get_default_library (
    DDS_Publisher * self )
```

<<*extension*>> (p. 1012) Gets the default XML library associated with a **DDS_Publisher** (p. 443).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

The default library or null if the default library was not set.

See also

DDS_Publisher_set_default_library (p. 452)

5.16.4.30 DDS_Publisher_get_default_profile()

```
const char * DDS_Publisher_get_default_profile (
    DDS_Publisher * self )
```

<<*extension*>> (p. 1012) Gets the default XML profile associated with a **DDS_Publisher** (p. 443).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

The default profile or null if the default profile was not set.

See also

DDS_Publisher_set_default_profile (p. 451)

5.16.4.31 DDS_Publisher_get_default_profile_library()

```
const char * DDS_Publisher_get_default_profile_library (
    DDS_Publisher * self )
```

<<*extension*>> (p. 1012) Gets the library where the default XML QoS profile is contained for a **DDS_Publisher** (p. 443).

The default profile library is automatically set when **DDS_Publisher_set_default_profile** (p. 451) is called.

This library can be different than the **DDS_Publisher** (p. 443) default library (see **DDS_Publisher_get_default_library** (p. 464)).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

The default profile library or null if the default profile was not set.

See also

DDS_Publisher_set_default_profile (p. 451)

5.16.4.32 DDS_Publisher_set_listener()

```
DDS_ReturnCode_t DDS_Publisher_set_listener (
    DDS_Publisher * self,
    const struct DDS_PublisherListener * l,
    DDS_StatusMask mask )
```

Sets the publisher listener.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>l</i>	<< <i>in</i> >> (p. 1013) DDS_PublisherListener (p. 1917) to set to.
<i>mask</i>	<< <i>in</i> >> (p. 1013) DDS_StatusMask (p. 1226) associated with the DDS_PublisherListener (p. 1917). The callback function on the listener cannot be NULL if the corresponding status is turned on in the <i>mask</i> .

Returns

One of the **Standard Return Codes** (p. 1221)

See also

set_listener (abstract) (p. 1362)

5.16.4.33 DDS_Publisher_get_listener()

```
struct DDS_PublisherListener DDS_Publisher_get_listener (
    DDS_Publisher * self )
```

Get the publisher listener.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_PublisherListener (p. 1917) of the **DDS_Publisher** (p. 443).

See also

DDS_Publisher_get_listenerX (p. 467)

get_listener (abstract) (p. 1363)

5.16.4.34 DDS_Publisher_get_listenerX()

```
DDS_ReturnCode_t DDS_Publisher_get_listenerX (
    DDS_Publisher * self,
    struct DDS_PublisherListener * listener )
```

<<*extension*>> (p. 1012) Get the publisher listener.

An alternative form of **get_listener** that fills in an existing listener structure rather than returning one on the stack.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>listener</i>	<< <i>inout</i> >> (p. 1013) DDS_PublisherListener (p. 1917) structure to be filled up. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_Publisher_get_listener (p. 466)

get_listener (abstract) (p. 1363)

5.16.4.35 DDS_Publisher_wait_for_acknowledgments()

```
DDS_ReturnCode_t DDS_Publisher_wait_for_acknowledgments (
    DDS_Publisher * self,
    const struct DDS_Duration_t * max_wait )
```

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 **DDS_DataReader** (p.615) 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 **DDS_Publisher** (p.443) 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 **DDS_DataWriter** (p.485) instances have **DDS_ReliabilityQosPolicy** (p.1936) kind set to RELIABLE, the operation will complete successfully.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>max_wait</i>	<< <i>in</i> >> (p. 1013) Specifies maximum time to wait for acknowledgements DDS_Duration_t (p. 1767) .

Returns

One of the **Standard Return Codes** (p.1221), **DDS_RETCODE_NOT_ENABLED** (p.1222), **DDS_RETCODE_↵_TIMEOUT** (p.1222)

5.16.4.36 DDS_Publisher_wait_for_asynchronous_publishing()

```
DDS_ReturnCode_t DDS_Publisher_wait_for_asynchronous_publishing (
    DDS_Publisher * self,
    const struct DDS_Duration_t * max_wait )
```

<<*extension*>> (p. 1012) 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 **DDS_DataWriter** (p.485) entities is sent and acknowledged (if reliable) by all matched **DDS_DataReader** (p.615) 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 **DDS_DataReader** (p.615) is complete in addition to what **DDS_Publisher_wait_for_acknowledgments** (p.467) provides.

If none of the **DDS_DataWriter** (p.485) instances have **DDS_PublishModeQosPolicy::kind** (p.1922) set to **DDS_↵_ASYNCHRONOUS_PUBLISH_MODE_QOS** (p.1319), the operation will complete immediately , with **DDS_↵_RETCODE_OK** (p.1222).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>max_wait</i>	<< <i>in</i> >> (p. 1013) Specifies maximum time to wait for acknowledgements DDS_Duration_t (p. 1767).

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_NOT_ENABLED** (p. 1222), **DDS_RETCODE_←_TIMEOUT** (p. 1222)

5.16.4.37 DDS_Publisher_lookup_datawriter_by_name()

```
DDS_DataWriter * DDS_Publisher_lookup_datawriter_by_name (
    DDS_Publisher * self,
    const char * datawriter_name )
```

<<*extension*>> (p. 1012) Retrieves a **DDS_DataWriter** (p. 485) contained within the **DDS_Publisher** (p. 443) the **DDS_DataWriter** (p. 485) entity name.

Every **DDS_DataWriter** (p. 485) in the system has an entity name which is configured and stored in the <<*extension*>> (p. 1012) EntityName policy, **ENTITY_NAME** (p. 1290).

This operation retrieves the **DDS_DataWriter** (p. 485) within the **DDS_Publisher** (p. 443) whose name matches the one specified. If there are several **DDS_DataWriter** (p. 485) with the same name within the **DDS_Publisher** (p. 443), the operation returns the first matching occurrence.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>datawriter_name</i>	<< <i>in</i> >> (p. 1013) Entity name of the DDS_DataWriter (p. 485).

Returns

The first **DDS_DataWriter** (p. 485) found with the specified name or NULL if it is not found.

See also

DDS_DomainParticipant_lookup_datawriter_by_name (p. 158)

5.16.5 Variable Documentation

5.16.5.1 DDS_DATAWRITER_QOS_PRINT_ALL

```
const struct DDS_DataWriterQos* DDS_DATAWRITER_QOS_PRINT_ALL [extern]
```

Special value which can be supplied to **DDS_DataWriterQos_to_string_w_params** (p. 534) indicating that all of the QoS should be printed.

The **DDS_DataWriterQos_to_string_w_params** (p. 534) API accepts a base QoS profile as one of its arguments. The resultant string only contains the differences with respect to the supplied base QoS profile. Supplying the **DDS_DATAWRITER_QOS_PRINT_ALL** (p. 469) sentinel value as the base QoS will result in all of the QoS being printed.

Note that there are some QoS policies and fields which are not intended for public use. Even when **DDS_DATAWRITER_QOS_PRINT_ALL** (p. 469) is supplied as the base, these will not be printed unless they differ from the documented default. If you want to see their values, you must use the `print_private` field within the **DDS_QosPrintFormat** (p. 1926) structure.

This value should only be used as the base parameter to the **DDS_DataWriterQos_to_string_w_params** (p. 534) API.

5.16.5.2 DDS_DATAWRITER_QOS_DEFAULT

```
const struct DDS_DataWriterQos DDS_DATAWRITER_QOS_DEFAULT [extern]
```

Special value for creating **DDS_DataWriter** (p. 485) with default QoS.

When used in **DDS_Publisher_create_datawriter** (p. 453), this special value is used to indicate that the **DDS_DataWriter** (p. 485) should be created with the default **DDS_DataWriter** (p. 485) QoS by means of the operation `get_default_datawriter_qos` and using the resulting QoS to create the **DDS_DataWriter** (p. 485).

When used in **DDS_Publisher_set_default_datawriter_qos** (p. 450), 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 **DDS_Publisher_set_default_datawriter_qos** (p. 450) operation had never been called.

When used in **DDS_DataWriter_set_qos** (p. 550), this special value is used to indicate that the QoS of the **DDS_DataWriter** (p. 485) should be changed to match the current default QoS set in the **DDS_Publisher** (p. 443) that the **DDS_DataWriter** (p. 485) 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 **DDS_DomainParticipant_get_default_datawriter_qos** (p. 91).

See also

DDS_Publisher_create_datawriter (p. 453)
DDS_Publisher_set_default_datawriter_qos (p. 450)
DDS_DataWriter_set_qos (p. 550)

Examples

HelloWorld_publisher.c.

5.16.5.3 DDS_DATAWRITER_QOS_USE_TOPIC_QOS

```
const struct DDS_DataWriterQos DDS_DATAWRITER_QOS_USE_TOPIC_QOS [extern]
```

Special value for creating **DDS_DataWriter** (p. 485) with a combination of the default **DDS_DataWriterQos** (p. 1666) and the **DDS_TopicQos** (p. 2042).

The use of this value is equivalent to the application obtaining the default **DDS_DataWriterQos** (p. 1666) and the **DDS_TopicQos** (p. 2042) (by means of the operation **DDS_Topic_get_qos** (p. 199)) and then combining these two QoS using the operation **DDS_Publisher_copy_from_topic_qos** (p. 462) whereby any policy that is set on the **DDS_TopicQos** (p. 2042) "overrides" the corresponding policy on the default QoS. The resulting QoS is then applied to the creation of the **DDS_DataWriter** (p. 485).

This value should only be used in **DDS_Publisher_create_datawriter** (p. 453).

See also

- DDS_Publisher_create_datawriter** (p. 453)
- DDS_Publisher_get_default_datawriter_qos** (p. 449)
- DDS_Topic_get_qos** (p. 199)
- DDS_Publisher_copy_from_topic_qos** (p. 462)

5.17 Data Writers

DDS_DataWriter (p. 485) entity and associated elements

Data Structures

- struct **FooDataWriter**
 - <<*interface*>> (p. 1013) <<*generic*>> (p. 1013) User data type specific data writer.
- struct **DDS_OfferedDeadlineMissedStatus**
 - DDS_OFFERED_DEADLINE_MISSED_STATUS** (p. 1228)
- struct **DDS_LivelinessLostStatus**
 - DDS_LIVELINESS_LOST_STATUS** (p. 1231)
- struct **DDS_OfferedIncompatibleQosStatus**
 - DDS_OFFERED_INCOMPATIBLE_QOS_STATUS** (p. 1229)
- struct **DDS_PublicationMatchedStatus**
 - DDS_PUBLICATION_MATCHED_STATUS** (p. 1232)
- struct **DDS_ServiceRequestAcceptedStatus**
 - DDS_SERVICE_REQUEST_ACCEPTED_STATUS** (p. 1233)
- struct **DDS_ReliableWriterCacheEventCount**
 - <<*extension*>> (p. 1012) 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. 1012) A summary of the state of a data writer's cache of unacknowledged samples written.
- struct **DDS_ReliableReaderActivityChangedStatus**

<<**extension**>> (p. 1012) Describes the activity (i.e. are acknowledgements forthcoming) of reliable readers matched to a reliable writer.

- struct **DDS_DataWriterCacheStatus**

<<**extension**>> (p. 1012) 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. 1012) 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 **DDS_DataWriter** (p. 485) entity.

- struct **DDS_DataWriterListener**

<<**interface**>> (p. 1013) **DDS_Listener** (p. 1816) for writer status.

Macros

- #define **DDS_OfferedDeadlineMissedStatus_INITIALIZER**

Initializer for new status instances.

- #define **DDS_LivelinessLostStatus_INITIALIZER**

Initializer for new status instances.

- #define **DDS_OfferedIncompatibleQoSStatus_INITIALIZER**

Initializer for new status instances.

- #define **DDS_PublicationMatchedStatus_INITIALIZER**

Initializer for new status instances.

- #define **DDS_ServiceRequestAcceptedStatus_INITIALIZER**

Initializer for new status instances.

- #define **DDS_ReliableWriterCacheChangedStatus_INITIALIZER**

Initializer for new status instances.

- #define **DDS_ReliableReaderActivityChangedStatus_INITIALIZER**

Initializer for new status instances.

- #define **DDS_DataWriterCacheStatus_INITIALIZER**

Initializer for new status instances.

- #define **DDS_DataWriterProtocolStatus_INITIALIZER**

Initializer for new status instances.

- #define **DDS_DataWriterQos_INITIALIZER**

Initializer for new QoS instances.

- #define **DDS_DataWriterListener_INITIALIZER**

Initializer for new **DDS_DataWriterListener** (p. 1644).

Typedefs

- typedef struct DDS_DataWriterImpl **DDS_DataWriter**
 <<**interface**>> (p. 1013) Allows an application to set the value of the data to be published under a given **DDS_Topic** (p. 175).
- typedef void(* **DDS_DataWriterListener_OfferedDeadlineMissedCallback**) (void *listener_data, **DDS_DataWriter** *writer, const struct **DDS_OfferedDeadlineMissedStatus** *status)
 Prototype of a **DDS_DataWriterListener** (p. 1644) on_offered_deadline_missed function.
- typedef void(* **DDS_DataWriterListener_LivelinessLostCallback**) (void *listener_data, **DDS_DataWriter** *writer, const struct **DDS_LivelinessLostStatus** *status)
 Prototype of a **DDS_DataWriterListener** (p. 1644) on_liveliness_lost function.
- typedef void(* **DDS_DataWriterListener_OfferedIncompatibleQosCallback**) (void *listener_data, **DDS_DataWriter** *writer, const struct **DDS_OfferedIncompatibleQosStatus** *status)
 Prototype of a **DDS_DataWriterListener** (p. 1644) on_offered_incompatible_qos function.
- typedef void(* **DDS_DataWriterListener_PublicationMatchedCallback**) (void *listener_data, **DDS_DataWriter** *writer, const struct **DDS_PublicationMatchedStatus** *status)
 Prototype of a **DDS_DataWriterListener** (p. 1644) on_publication_matched function.
- typedef void(* **DDS_DataWriterListener_ReliableWriterCacheChangedCallback**) (void *listener_data, **DDS_DataWriter** *writer, const struct **DDS_ReliableWriterCacheChangedStatus** *status)
 <<**extension**>> (p. 1012) A change has occurred in the writer's cache.
- typedef void(* **DDS_DataWriterListener_ReliableReaderActivityChangedCallback**) (void *listener_data, **DDS_DataWriter** *writer, const struct **DDS_ReliableReaderActivityChangedStatus** *status)
 <<**extension**>> (p. 1012) A matched reliable reader has become active or become inactive.
- typedef void(* **DDS_DataWriterListener_SampleRemovedCallback**) (void *listener_data, **DDS_DataWriter** *writer, const struct **DDS_Cookie_t** *cookie)
 <<**extension**>> (p. 1012) A sample has been removed from the DataWriter's queue.
- typedef void(* **DDS_DataWriterListener_InstanceReplacedCallback**) (void *listener_data, **DDS_DataWriter** *writer, const **DDS_InstanceHandle_t** *handle)
 <<**extension**>> (p. 1012) An instance previously registered with the writer has been replaced
- typedef void(* **DDS_DataWriterListener_OnApplicationAcknowledgmentCallback**) (void *listener_data, **DDS_DataWriter** *writer, const struct **DDS_AcknowledgmentInfo** *info)
 <<**extension**>> (p. 1012) An application-level acknowledgment has been received for a sample
- typedef void(* **DDS_DataWriterListener_ServiceRequestAcceptedCallback**) (void *listener_data, **DDS_DataWriter** *writer, const struct **DDS_ServiceRequestAcceptedStatus** *status)
 <<**extension**>> (p. 1012) a **DDS_ServiceRequest** (p. 2001) for the **DDS_TopicQuery** (p. 707) service is dispatched to this **DDS_DataWriter** (p. 485).

Functions

- **FooDataWriter * FooDataWriter_narrow** (**DDS_DataWriter** *writer)
 Narrow the given **DDS_DataWriter** (p. 485) pointer to a **FooDataWriter** (p. 2113) pointer.
- **DDS_DataWriter * FooDataWriter_as_datawriter** (**FooDataWriter** *writer)
 Widen the given **FooDataWriter** (p. 2113) pointer to a **DDS_DataWriter** (p. 485) pointer.
- **DDS_InstanceHandle_t FooDataWriter_register_instance** (**FooDataWriter** *self, const **Foo** *instance_data)
 Informs RTI Connexx that the application will be modifying a particular instance.
- **DDS_InstanceHandle_t FooDataWriter_register_instance_w_timestamp** (**FooDataWriter** *self, const **Foo** *instance_data, const struct **DDS_Time_t** *source_timestamp)

Performs the same functions as `register_instance` except that the application provides the value for the `source_timestamp`.

- **DDS_InstanceHandle_t FooDataWriter_register_instance_w_params** (FooDataWriter *self, const Foo *instance_data, struct DDS_WriteParams_t *params)

Performs the same function as `FooDataWriter_register_instance` (p. 491) and `FooDataWriter_register_instance_w_timestamp` (p. 492) except that it also provides the values contained in `params`.

- **DDS_ReturnCode_t FooDataWriter_unregister_instance** (FooDataWriter *self, const Foo *instance_data, const DDS_InstanceHandle_t *handle)

Reverses the action of `FooDataWriter_register_instance` (p. 491).

- **DDS_ReturnCode_t FooDataWriter_unregister_instance_w_timestamp** (FooDataWriter *self, const Foo *instance_data, const DDS_InstanceHandle_t *handle, const struct DDS_Time_t *source_timestamp)

Performs the same function as `FooDataWriter_unregister_instance` (p. 493) except that it also provides the value for the `source_timestamp`.

- **DDS_ReturnCode_t FooDataWriter_unregister_instance_w_params** (FooDataWriter *self, const Foo *instance_data, struct DDS_WriteParams_t *params)

Performs the same function as `FooDataWriter_unregister_instance` (p. 493) and `FooDataWriter_unregister_instance_w_timestamp` (p. 495) except that it also provides the values contained in `params`.

- **DDS_ReturnCode_t FooDataWriter_write** (FooDataWriter *self, const Foo *instance_data, const DDS_InstanceHandle_t *handle)

Modifies the value of a data instance.

- **DDS_ReturnCode_t FooDataWriter_write_w_timestamp** (FooDataWriter *self, const Foo *instance_data, const DDS_InstanceHandle_t *handle, const struct DDS_Time_t *source_timestamp)

Performs the same function as `FooDataWriter_write` (p. 496) except that it also provides the value for the `source_timestamp`.

- **DDS_ReturnCode_t FooDataWriter_write_w_params** (FooDataWriter *self, const Foo *instance_data, struct DDS_WriteParams_t *params)

Performs the same function as `FooDataWriter_write` (p. 496) and `FooDataWriter_write_w_timestamp` (p. 500) except that it also provides the values contained in `params`.

- **DDS_ReturnCode_t FooDataWriter_dispose** (FooDataWriter *self, const Foo *instance_data, const DDS_InstanceHandle_t *instance_handle)

Requests the middleware to delete the instance.

- **DDS_ReturnCode_t FooDataWriter_dispose_w_timestamp** (FooDataWriter *self, const Foo *instance_data, const DDS_InstanceHandle_t *instance_handle, const struct DDS_Time_t *source_timestamp)

Performs the same functions as `dispose` except that the application provides the value for the `source_timestamp` that is made available to `DDS_DataReader` (p. 615) objects by means of the `source_timestamp` attribute inside the `DDS_SampleInfo` (p. 1985).

- **DDS_ReturnCode_t FooDataWriter_dispose_w_params** (FooDataWriter *self, const Foo *instance_data, struct DDS_WriteParams_t *params)

Performs the same function as `FooDataWriter_dispose` (p. 502) and `FooDataWriter_dispose_w_timestamp` (p. 503) except that it also provides the values contained in `params`.

- **DDS_ReturnCode_t FooDataWriter_get_key_value** (FooDataWriter *self, Foo *key_holder, const DDS_InstanceHandle_t *handle)

Retrieve the instance `key` that corresponds to an instance `handle`.

- **DDS_InstanceHandle_t FooDataWriter_lookup_instance** (FooDataWriter *self, const Foo *key_holder)

Retrieve the instance `handle` that corresponds to an instance `key_holder`.

- **void * FooDataWriter_create_data** (FooDataWriter *self)

Creates a data sample and initializes it.

- **void * FooDataWriter_create_data_w_params** (FooDataWriter *self, const struct DDS_TypeAllocationParams_t *alloc_params)

Creates a data sample and initializes it.

- **DDS_Boolean FooDataWriter_delete_data** (FooDataWriter *self, Foo *sample)
Destroys a user data type instance.
- **void * FooDataWriter_delete_data_w_params** (FooDataWriter *self, Foo *sample, const struct DDS_↵
TypeDeallocationParams_t *dealloc_params)
Destroys a user data type instance.
- **DDS_ReturnCode_t FooDataWriter_get_loan** (FooDataWriter *self, Foo **sample)
Gets a sample managed by the DataWriter.
- **DDS_ReturnCode_t FooDataWriter_discard_loan** (FooDataWriter *self, Foo *sample)
Returns a loaned sample back to the DataWriter.
- **DDS_ReturnCode_t DDS_OfferedDeadlineMissedStatus_initialize** (struct DDS_OfferedDeadlineMissed↵
Status *self)
Initializer for new status instances.
- **DDS_ReturnCode_t DDS_OfferedDeadlineMissedStatus_copy** (struct DDS_OfferedDeadlineMissed↵
Status *self, const struct DDS_OfferedDeadlineMissedStatus *source)
Copy the contents of the given status into this status.
- **DDS_ReturnCode_t DDS_OfferedDeadlineMissedStatus_finalize** (struct DDS_OfferedDeadlineMissed↵
Status *self)
Free any dynamic memory allocated by status instances.
- **DDS_Boolean DDS_OfferedDeadlineMissedStatus_equals** (const struct DDS_OfferedDeadlineMissed↵
Status *left, const struct DDS_OfferedDeadlineMissedStatus *right)
Compares two DDS_OfferedDeadlineMissedStatus (p. 1861) for equality.
- **DDS_ReturnCode_t DDS_LivelinessLostStatus_initialize** (struct DDS_LivelinessLostStatus *self)
Initializer for new status instances.
- **DDS_ReturnCode_t DDS_LivelinessLostStatus_copy** (struct DDS_LivelinessLostStatus *self, const struct
DDS_LivelinessLostStatus *source)
Copy the contents of the given status into this status.
- **DDS_ReturnCode_t DDS_LivelinessLostStatus_finalize** (struct DDS_LivelinessLostStatus *self)
Free any dynamic memory allocated by status instances.
- **DDS_Boolean DDS_LivelinessLostStatus_equals** (const struct DDS_LivelinessLostStatus *left, const
struct DDS_LivelinessLostStatus *right)
Compares two DDS_LivelinessLostStatus (p. 1822) for equality.
- **DDS_ReturnCode_t DDS_OfferedIncompatibleQosStatus_initialize** (struct DDS_OfferedIncompatible↵
QosStatus *self)
Initializer for new status instances.
- **DDS_ReturnCode_t DDS_OfferedIncompatibleQosStatus_copy** (struct DDS_OfferedIncompatibleQos↵
Status *self, const struct DDS_OfferedIncompatibleQosStatus *source)
Copy the contents of the given status into this status.
- **DDS_ReturnCode_t DDS_OfferedIncompatibleQosStatus_finalize** (struct DDS_OfferedIncompatible↵
QosStatus *self)
Free any dynamic memory allocated by status instances.
- **DDS_Boolean DDS_OfferedIncompatibleQosStatus_equals** (const struct DDS_OfferedIncompatible↵
QosStatus *left, const struct DDS_OfferedIncompatibleQosStatus *right)
Compares two DDS_OfferedIncompatibleQosStatus (p. 1862) for equality.
- **DDS_ReturnCode_t DDS_PublicationMatchedException_initialize** (struct DDS_PublicationMatchedException
*self)
Initializer for new status instances.
- **DDS_ReturnCode_t DDS_PublicationMatchedException_copy** (struct DDS_PublicationMatchedException *self,
const struct DDS_PublicationMatchedException *source)
Copy the contents of the given status into this status.

- **DDS_ReturnCode_t DDS_PublicationMatchedStatus_finalize** (struct **DDS_PublicationMatchedStatus** *self)
Free any dynamic memory allocated by status instances.
- **DDS_Boolean DDS_PublicationMatchedStatus_equals** (const struct **DDS_PublicationMatchedStatus** *left, const struct **DDS_PublicationMatchedStatus** *right)
*Compares two **DDS_PublicationMatchedStatus** (p. 1914) for equality.*
- **DDS_ReturnCode_t DDS_ServiceRequestAcceptedStatus_initialize** (struct **DDS_ServiceRequestAcceptedStatus** *self)
Initializer for new status instances.
- **DDS_ReturnCode_t DDS_ServiceRequestAcceptedStatus_copy** (struct **DDS_ServiceRequestAcceptedStatus** *self, const struct **DDS_ServiceRequestAcceptedStatus** *source)
Copy the contents of the given status into this status.
- **DDS_ReturnCode_t DDS_ServiceRequestAcceptedStatus_finalize** (struct **DDS_ServiceRequestAcceptedStatus** *self)
Free any dynamic memory allocated by status instances.
- **DDS_Boolean DDS_ServiceRequestAcceptedStatus_equals** (const struct **DDS_ServiceRequestAcceptedStatus** *left, const struct **DDS_ServiceRequestAcceptedStatus** *right)
*Compares two **DDS_ServiceRequestAcceptedStatus** (p. 2002) for equality.*
- **DDS_Boolean DDS_ReliableWriterCacheEventCount_equals** (const struct **DDS_ReliableWriterCacheEventCount** *left, const struct **DDS_ReliableWriterCacheEventCount** *right)
*Compares two **DDS_ReliableWriterCacheEventCount** (p. 1945) for equality.*
- **DDS_ReturnCode_t DDS_ReliableWriterCacheChangedStatus_initialize** (struct **DDS_ReliableWriterCacheChangedStatus** *self)
Initializer for new status instances.
- **DDS_ReturnCode_t DDS_ReliableWriterCacheChangedStatus_copy** (struct **DDS_ReliableWriterCacheChangedStatus** *self, const struct **DDS_ReliableWriterCacheChangedStatus** *source)
Copy the contents of the given status into this status.
- **DDS_ReturnCode_t DDS_ReliableWriterCacheChangedStatus_finalize** (struct **DDS_ReliableWriterCacheChangedStatus** *self)
Free any dynamic memory allocated by status instances.
- **DDS_Boolean DDS_ReliableWriterCacheChangedStatus_equals** (const struct **DDS_ReliableWriterCacheChangedStatus** *left, const struct **DDS_ReliableWriterCacheChangedStatus** *right)
*Compares two **DDS_ReliableWriterCacheChangedStatus** (p. 1942) for equality.*
- **DDS_ReturnCode_t DDS_ReliableReaderActivityChangedStatus_initialize** (struct **DDS_ReliableReaderActivityChangedStatus** *self)
Initializer for new status instances.
- **DDS_ReturnCode_t DDS_ReliableReaderActivityChangedStatus_copy** (struct **DDS_ReliableReaderActivityChangedStatus** *self, const struct **DDS_ReliableReaderActivityChangedStatus** *source)
Copy the contents of the given status into this status.
- **DDS_ReturnCode_t DDS_ReliableReaderActivityChangedStatus_finalize** (struct **DDS_ReliableReaderActivityChangedStatus** *self)
Free any dynamic memory allocated by status instances.
- **DDS_Boolean DDS_ReliableReaderActivityChangedStatus_equals** (const struct **DDS_ReliableReaderActivityChangedStatus** *left, const struct **DDS_ReliableReaderActivityChangedStatus** *right)
*Compares two **DDS_ReliableReaderActivityChangedStatus** (p. 1940) for equality.*
- **DDS_ReturnCode_t DDS_DataWriterCacheStatus_initialize** (struct **DDS_DataWriterCacheStatus** *self)
Initializer for new status instances.
- **DDS_ReturnCode_t DDS_DataWriterCacheStatus_copy** (struct **DDS_DataWriterCacheStatus** *self, const struct **DDS_DataWriterCacheStatus** *source)

Copy the contents of the given status into this status.

- **DDS_ReturnCode_t DDS_DataWriterCacheStatus_finalize** (struct **DDS_DataWriterCacheStatus** *self)

Free any dynamic memory allocated by status instances.

- **DDS_Boolean DDS_DataWriterCacheStatus_equals** (const struct **DDS_DataWriterCacheStatus** *left, const struct **DDS_DataWriterCacheStatus** *right)

*Compares two **DDS_DataWriterCacheStatus** (p. 1641) for equality.*

- **DDS_ReturnCode_t DDS_DataWriterProtocolStatus_initialize** (struct **DDS_DataWriterProtocolStatus** *self)

Initializer for new status instances.

- **DDS_ReturnCode_t DDS_DataWriterProtocolStatus_copy** (struct **DDS_DataWriterProtocolStatus** *self, const struct **DDS_DataWriterProtocolStatus** *source)

Copy the contents of the given status into this status.

- **DDS_ReturnCode_t DDS_DataWriterProtocolStatus_finalize** (struct **DDS_DataWriterProtocolStatus** *self)

Free any dynamic memory allocated by status instances.

- **DDS_Boolean DDS_DataWriterProtocolStatus_equals** (const struct **DDS_DataWriterProtocolStatus** *left, const struct **DDS_DataWriterProtocolStatus** *right)

*Compares two **DDS_DataWriterProtocolStatus** (p. 1654) for equality.*

- **DDS_Boolean DDS_DataWriterQos_equals** (const struct **DDS_DataWriterQos** *self, const struct **DDS_DataWriterQos** *other)

*Compares two **DDS_DataWriterQos** (p. 1666) for equality.*

- **DDS_ReturnCode_t DDS_DataWriterQos_print** (const struct **DDS_DataWriterQos** *self)

*Prints this **DDS_DataWriterQos** (p. 1666) to stdout.*

- **DDS_ReturnCode_t DDS_DataWriterQos_to_string** (const struct **DDS_DataWriterQos** *self, char *string, **DDS_UnsignedLong** *string_size)

*Obtains a string representation of this **DDS_DataWriterQos** (p. 1666).*

- **DDS_ReturnCode_t DDS_DataWriterQos_to_string_w_params** (const struct **DDS_DataWriterQos** *self, char *string, **DDS_UnsignedLong** *string_size, const struct **DDS_DataWriterQos** *base, const struct **DDS_QosPrintFormat** *format)

*Obtains a string representation of this **DDS_DataWriterQos** (p. 1666).*

- **DDS_ReturnCode_t DDS_DataWriterQos_initialize** (struct **DDS_DataWriterQos** *self)

Initializer for new QoS instances.

- **DDS_ReturnCode_t DDS_DataWriterQos_finalize** (struct **DDS_DataWriterQos** *self)

*Free any dynamic memory allocated by the policies in this **DDS_DataWriterQos** (p. 1666).*

- **DDS_ReturnCode_t DDS_DataWriterQos_copy** (struct **DDS_DataWriterQos** *self, const struct **DDS_DataWriterQos** *source)

Copy the contents of the given QoS into this QoS.

- **DDS_Entity * DDS_DataWriter_as_entity** (**DDS_DataWriter** *dataWriter)

*Access a **DDS_DataWriter** (p. 485)'s supertype instance.*

- **DDS_ReturnCode_t DDS_DataWriter_assert_liveliness** (**DDS_DataWriter** *self)

*This operation manually asserts the liveliness of this **DDS_DataWriter** (p. 485).*

- **DDS_ReturnCode_t DDS_DataWriter_get_matched_subscription_locators** (**DDS_DataWriter** *self, struct **DDS_LocatorSeq** *locators)

*<<extension>> (p. 1012) Retrieve the list of locators for subscriptions currently "associated" with this **DDS_DataWriter** (p. 485).*

- **DDS_ReturnCode_t DDS_DataWriter_get_matched_subscriptions** (**DDS_DataWriter** *self, struct **DDS_InstanceHandleSeq** *subscription_handles)

*Retrieve the list of subscriptions currently "associated" with this **DDS_DataWriter** (p. 485).*

- **DDS_ReturnCode_t DDS_DataWriter_is_matched_subscription_active** (**DDS_DataWriter** *self, **DDS_Boolean** *is_active, const **DDS_InstanceHandle_t** *subscription_handle)

Check if a subscription currently matched with a *DataWriter* is active.

- **DDS_ReturnCode_t DDS_DataWriter_get_matched_subscription_data** (**DDS_DataWriter** *self, struct **DDS_SubscriptionBuiltinTopicData** *subscription_data, const **DDS_InstanceHandle_t** *subscription_handle)

This operation retrieves the information on a subscription that is currently "associated" with the *DDS_DataWriter* (p. 485).

- **DDS_ReturnCode_t DDS_DataWriter_get_matched_subscription_participant_data** (**DDS_DataWriter** *self, struct **DDS_ParticipantBuiltinTopicData** *participant_data, const **DDS_InstanceHandle_t** *subscription_handle)

This operation retrieves the information on the discovered *DDS_DomainParticipant* (p. 77) associated with the subscription that is currently matching with the *DDS_DataWriter* (p. 485).

- **DDS_Topic * DDS_DataWriter_get_topic** (**DDS_DataWriter** *self)

This operation returns the *DDS_Topic* (p. 175) associated with the *DDS_DataWriter* (p. 485).

- **DDS_Publisher * DDS_DataWriter_get_publisher** (**DDS_DataWriter** *self)

This operation returns the *DDS_Publisher* (p. 443) to which the *DDS_DataWriter* (p. 485) belongs.

- **DDS_ReturnCode_t DDS_DataWriter_wait_for_acknowledgments** (**DDS_DataWriter** *self, const struct **DDS_Duration_t** *max_wait)

Blocks the calling thread until all data written by reliable *DDS_DataWriter* (p. 485) entity is acknowledged, or until timeout expires.

- **DDS_ReturnCode_t DDS_DataWriter_is_sample_app_acknowledged** (**DDS_DataWriter** *self, **DDS_Boolean** *is_app_ack, const struct **DDS_SampleIdentity_t** *identity)

This function can be used to see if a sample has been application acknowledged.

- **DDS_ReturnCode_t DDS_DataWriter_wait_for_asynchronous_publishing** (**DDS_DataWriter** *self, const struct **DDS_Duration_t** *max_wait)

<<extension>> (p. 1012) Blocks the calling thread until asynchronous sending is complete.

- **DDS_ReturnCode_t DDS_DataWriter_get_liveliness_lost_status** (**DDS_DataWriter** *self, struct **DDS_LivelinessLostStatus** *status)

Accesses the *DDS_LIVELINESS_LOST_STATUS* (p. 1231) communication status.

- **DDS_ReturnCode_t DDS_DataWriter_get_offered_deadline_missed_status** (**DDS_DataWriter** *self, struct **DDS_OfferedDeadlineMissedStatus** *status)

Accesses the *DDS_OFFERED_DEADLINE_MISSED_STATUS* (p. 1228) communication status.

- **DDS_ReturnCode_t DDS_DataWriter_get_offered_incompatible_qos_status** (**DDS_DataWriter** *self, struct **DDS_OfferedIncompatibleQosStatus** *status)

Accesses the *DDS_OFFERED_INCOMPATIBLE_QOS_STATUS* (p. 1229) communication status.

- **DDS_ReturnCode_t DDS_DataWriter_get_publication_matched_status** (**DDS_DataWriter** *self, struct **DDS_PublicationMatchedStatus** *status)

Accesses the *DDS_PUBLICATION_MATCHED_STATUS* (p. 1232) communication status.

- **DDS_ReturnCode_t DDS_DataWriter_get_reliable_writer_cache_changed_status** (**DDS_DataWriter** *self, struct **DDS_ReliableWriterCacheChangedStatus** *status)

<<extension>> (p. 1012) Get the reliable cache status for this writer.

- **DDS_ReturnCode_t DDS_DataWriter_get_reliable_reader_activity_changed_status** (**DDS_DataWriter** *self, struct **DDS_ReliableReaderActivityChangedStatus** *status)

<<extension>> (p. 1012) Get the reliable reader activity changed status for this writer.

- **DDS_ReturnCode_t DDS_DataWriter_get_datawriter_cache_status** (**DDS_DataWriter** *self, struct **DDS_DataWriterCacheStatus** *status)

<<extension>> (p. 1012) Get the datawriter cache status for this writer.

- **DDS_ReturnCode_t DDS_DataWriter_get_datawriter_protocol_status** (**DDS_DataWriter** *self, struct **DDS_DataWriterProtocolStatus** *status)

<<extension>> (p. 1012) Get the datawriter protocol status for this writer.

- **DDS_ReturnCode_t DDS_DataWriter_get_matched_subscription_datawriter_protocol_status** (**DDS_DataWriter** *self, struct **DDS_DataWriterProtocolStatus** *status, const **DDS_InstanceHandle_t** *subscription_handle)

- <<*extension*>> (p. 1012) Get the datawriter protocol status for this writer, per matched subscription identified by the *subscription_handle*.
- **DDS_ReturnCode_t DDS_DataWriter_get_matched_subscription_datawriter_protocol_status_by_**↵
locator (**DDS_DataWriter** *self, struct **DDS_DataWriterProtocolStatus** *status, const struct **DDS_Locator_t** *locator)

 <<*extension*>> (p. 1012) Get the datawriter protocol status for this writer, per matched subscription identified by the *locator*.
 - **DDS_ReturnCode_t DDS_DataWriter_get_service_request_accepted_status** (**DDS_DataWriter** *self, struct **DDS_ServiceRequestAcceptedStatus** *status)

 Accesses the **DDS_SERVICE_REQUEST_ACCEPTED_STATUS** (p. 1233) communication status.
 - **DDS_ReturnCode_t DDS_DataWriter_set_qos** (**DDS_DataWriter** *self, const struct **DDS_DataWriterQos** *qos)

 Sets the writer QoS.
 - **DDS_ReturnCode_t DDS_DataWriter_set_qos_with_profile** (**DDS_DataWriter** *self, const char *library_↵
name, const char *profile_name)

 <<*extension*>> (p. 1012) Change the QoS of this writer using the input XML QoS profile.
 - **DDS_ReturnCode_t DDS_DataWriter_get_qos** (**DDS_DataWriter** *self, struct **DDS_DataWriterQos** *qos)

 Gets the writer QoS.
 - **DDS_ReturnCode_t DDS_DataWriter_set_property** (**DDS_DataWriter** *self, const char *property_name, const char *value, **DDS_Boolean** propagate)

 Set the value for a property that applies to a DataWriter.
 - **DDS_ReturnCode_t DDS_DataWriter_set_listener** (**DDS_DataWriter** *self, const struct **DDS_DataWriter**↵
Listener *l, **DDS_StatusMask** mask)

 Sets the writer listener.
 - struct **DDS_DataWriterListener** **DDS_DataWriter_get_listener** (**DDS_DataWriter** *self)

 Get the writer listener.
 - **DDS_ReturnCode_t DDS_DataWriter_get_listenerX** (**DDS_DataWriter** *self, struct **DDS_DataWriter**↵
Listener *listener)

 <<*extension*>> (p. 1012) Get the writer listener.
 - **DDS_ReturnCode_t DDS_DataWriter_flush** (**DDS_DataWriter** *self)

 <<*extension*>> (p. 1012) Flushes the batch in progress in the context of the calling thread.
 - **DDS_ReturnCode_t DDS_DataWriter_take_discovery_snapshot** (**DDS_DataWriter** *self, const char *file_↵
_name)

 Take a snapshot of the compatible and incompatible remote readers matched by a local writer.

5.17.1 Detailed Description

DDS_DataWriter (p. 485) entity and associated elements

5.17.2 Macro Definition Documentation

5.17.2.1 DDS_OfferedDeadlineMissedStatus_INITIALIZER

```
#define DDS_OfferedDeadlineMissedStatus_INITIALIZER
```

Initializer for new status instances.

New **DDS_OfferedDeadlineMissedStatus** (p. 1861) instances stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those fields that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a status structure is to initialize it on the stack at the time of its creation. **DDS_OfferedDeadlineMissedStatus_finalize** (p. 512) should be called to free the contained fields that use dynamic memory:

```
struct DDS_OfferedDeadlineMissedStatus myStatus = DDS_OfferedDeadlineMissedStatus_INITIALIZER;
```

```
DDS_DataWriter_get_offered_deadline_missed_status(myDataWriter, &myStatus);
```

```
DDS_OfferedDeadlineMissedStatus_finalize(&myStatus);
```

See also

DDS_OfferedDeadlineMissedStatus_initialize (p. 510)

DDS_DataWriter_get_offered_deadline_missed_status (p. 546)

DDS_OfferedDeadlineMissedStatus_finalize (p. 512)

5.17.2.2 DDS_LivelinessLostStatus_INITIALIZER

```
#define DDS_LivelinessLostStatus_INITIALIZER
```

Initializer for new status instances.

New **DDS_LivelinessLostStatus** (p. 1822) instances stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those fields that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a status structure is to initialize it on the stack at the time of its creation. **DDS_LivelinessLostStatus_finalize** (p. 514) should be called to free the contained fields that use dynamic memory:

```
struct DDS_LivelinessLostStatus myStatus = DDS_LivelinessLostStatus_INITIALIZER;
```

```
DDS_DataWriter_get_liveliness_lost_status(myDataWriter, &myStatus);
```

```
DDS_LivelinessLostStatus_finalize(&myStatus);
```

See also

DDS_LivelinessLostStatus_initialize (p. 513)

DDS_DataWriter_get_liveliness_lost_status (p. 545)

DDS_LivelinessLostStatus_finalize (p. 514)

5.17.2.3 DDS_OfferedIncompatibleQosStatus_INITIALIZER

```
#define DDS_OfferedIncompatibleQosStatus_INITIALIZER
```

Initializer for new status instances.

New **DDS_OfferedIncompatibleQosStatus** (p. 1862) instances stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those fields that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a status structure is to initialize it on the stack at the time of its creation. **DDS_OfferedIncompatibleQosStatus_finalize** (p. 516) should be called to free the contained fields that use dynamic memory:

```
struct DDS_OfferedIncompatibleQosStatus myStatus = DDS_OfferedIncompatibleQosStatus_INITIALIZER;
```

```
DDS_DataWriter_get_offered_incompatible_qos_status(myDataWriter, &myStatus);
```

```
DDS_OfferedIncompatibleQosStatus_finalize(&myStatus);
```

See also

DDS_DataWriter_get_offered_incompatible_qos_status (p. 546)

DDS_OfferedIncompatibleQosStatus_finalize (p. 516)

5.17.2.4 DDS_PublicationMatchedException_INITIALIZER

```
#define DDS_PublicationMatchedException_INITIALIZER
```

Initializer for new status instances.

New **DDS_PublicationMatchedException** (p. 1914) instances stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those fields that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a status structure is to initialize it on the stack at the time of its creation. **DDS_PublicationMatchedException_finalize** (p. 519) should be called to free the contained fields that use dynamic memory:

```
struct DDS_PublicationMatchedException myStatus = DDS_PublicationMatchedException_INITIALIZER;
```

```
DDS_DataWriter_get_publication_matched_status(myDataWriter, &myStatus);
```

```
DDS_PublicationMatchedException_finalize(&myStatus);
```

See also

DDS_DataWriter_get_publication_matched_status (p. 546)

DDS_PublicationMatchedException_finalize (p. 519)

5.17.2.5 DDS_ServiceRequestAcceptedStatus_INITIALIZER

```
#define DDS_ServiceRequestAcceptedStatus_INITIALIZER
```

Initializer for new status instances.

New **DDS_ServiceRequestAcceptedStatus** (p. 2002) instances stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those fields that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a status structure is to initialize it on the stack at the time of its creation. **DDS_ServiceRequestAcceptedStatus_finalize** (p. 521) should be called to free the contained fields that use dynamic memory:

```
struct DDS_ServiceRequestAcceptedStatus myStatus = DDS_ServiceRequestAcceptedStatus_INITIALIZER;
```

```
DDS_DataWriter_get_service_request_accepted_status(myDataWriter, &myStatus);
```

```
DDS_ServiceRequestAcceptedStatus_finalize(&myStatus);
```

See also

DDS_DataWriter_get_service_request_accepted_status (p. 550)

DDS_ServiceRequestAcceptedStatus_finalize (p. 521)

5.17.2.6 DDS_ReliableWriterCacheChangedStatus_INITIALIZER

```
#define DDS_ReliableWriterCacheChangedStatus_INITIALIZER
```

Initializer for new status instances.

New **DDS_ReliableWriterCacheChangedStatus** (p. 1942) instances stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those fields that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a status structure is to initialize it on the stack at the time of its creation. **DDS_ReliableWriterCacheChangedStatus_finalize** (p. 524) should be called to free the contained fields that use dynamic memory:

```
struct DDS_ReliableWriterCacheChangedStatus myStatus = DDS_ReliableWriterCacheChangedStatus_INITIALIZER;
```

```
DDS_DataWriter_get_reliable_writer_cache_changed_status(myDataWriter, &myStatus);
```

```
DDS_ReliableWriterCacheChangedStatus_finalize(&myStatus);
```

See also

DDS_DataWriter_get_reliable_writer_cache_changed_status (p. 547)

DDS_ReliableWriterCacheChangedStatus_finalize (p. 524)

5.17.2.7 DDS_ReliableReaderActivityChangedStatus_INITIALIZER

```
#define DDS_ReliableReaderActivityChangedStatus_INITIALIZER
```

Initializer for new status instances.

New **DDS_ReliableReaderActivityChangedStatus** (p. 1940) instances stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those fields that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a status structure is to initialize it on the stack at the time of its creation. **DDS_ReliableReaderActivityChangedStatus_finalize** (p. 526) should be called to free the contained fields that use dynamic memory:

```
struct DDS_ReliableReaderActivityChangedStatus myStatus = DDS_ReliableReaderActivityChangedStatus_INITIALIZER;
DDS_DataWriter_get_reliable_reader_activity_changed_status(myDataWriter, &myStatus);
DDS_ReliableReaderActivityChangedStatus_finalize(&myStatus);
```

See also

DDS_DataWriter_get_reliable_reader_activity_changed_status (p. 547)

DDS_ReliableReaderActivityChangedStatus_finalize (p. 526)

5.17.2.8 DDS_DataWriterCacheStatus_INITIALIZER

```
#define DDS_DataWriterCacheStatus_INITIALIZER
```

Initializer for new status instances.

New **DDS_DataWriterCacheStatus** (p. 1641) instances stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those fields that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a status structure is to initialize it on the stack at the time of its creation. **DDS_DataWriterCacheStatus_finalize** (p. 529) should be called to free the contained fields that use dynamic memory:

```
struct DDS_DataWriterCacheStatus myStatus = DDS_DataWriterCacheStatus_INITIALIZER;
DDS_DataWriter_get_datawriter_cache_status(myDataWriter, &myStatus);
DDS_DataWriterCacheStatus_finalize(&myStatus);
```

See also

DDS_DataWriter_get_datawriter_cache_status (p. 548)

DDS_DataWriterCacheStatus_finalize (p. 529)

5.17.2.9 DDS_DataWriterProtocolStatus_INITIALIZER

```
#define DDS_DataWriterProtocolStatus_INITIALIZER
```

Initializer for new status instances.

New **DDS_DataWriterProtocolStatus** (p. 1654) instances stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those fields that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a status structure is to initialize it on the stack at the time of its creation. **DDS_DataWriterProtocolStatus_finalize** (p. 532) should be called to free the contained fields that use dynamic memory:

```
struct DDS_DataWriterProtocolStatus myStatus = DDS_DataWriterProtocolStatus_INITIALIZER;
```

```
DDS_DataWriter_get_datawriter_protocol_status(myDataWriter, &myStatus);
```

```
DDS_DataWriterProtocolStatus_finalize(&myStatus);
```

See also

DDS_DataWriter_get_datawriter_protocol_status (p. 548)

DDS_DataWriterProtocolStatus_finalize (p. 532)

5.17.2.10 DDS_DataWriterQos_INITIALIZER

```
#define DDS_DataWriterQos_INITIALIZER
```

Initializer for new QoS instances.

New **DDS_DataWriterQos** (p. 1666) instances stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those contained QoS policies that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a new QoS structure is to initialize it on the stack at the time of its creation:

```
struct DDS_DataWriterQos myQos = DDS_DataWriterQos_INITIALIZER;
```

Note that the above assignment is not a substitute for calling **DDS_Publisher_get_default_datawriter_qos** (p. 449) or **DDS_DataWriter_get_qos** (p. 552); one of those should be called subsequently to setting the QoS of a new or existing entity. **DDS_DataWriterQos_finalize** (p. 536) should be called to free the contained QoS policies that use dynamic memory:

```
struct DDS_DataWriterQos myQos = DDS_DataWriterQos_INITIALIZER;
```

```
DDS_Publisher_get_default_datawriter_qos(myPub, &myQos);
```

```
DDS_DataWriter_set_qos(myDataWriter, &myQos);
```

```
DDS_DataWriterQos_finalize(&myQos);
```

See also

DDS_Publisher_get_default_datawriter_qos (p. 449)

DDS_DataWriterQos_finalize (p. 536)

5.17.2.11 DDS_DataWriterListener_INITIALIZER

```
#define DDS_DataWriterListener_INITIALIZER
```

Initializer for new **DDS_DataWriterListener** (p. 1644).

All the new **DDS_DataWriterListener** (p. 1644) instances allocated in the stack should be initialized to this value. No memory is allocated.

```
struct DDS_DataWriterListener listener = DDS_DataWriterListener_INITIALIZER;

/* initialize listener functions */

listener.on_offered_deadline_missed = ....;

DDS_DataWriter_set_listener(myDataWriter, &listener, mask);
```

See also

DDS_DataWriter_set_listener (p. 553)

DDS_DataWriterListener (p. 1644)

5.17.3 Typedef Documentation

5.17.3.1 DDS_DataWriter

```
typedef struct DDS_DataWriterImpl DDS_DataWriter
```

<<*interface*>> (p. 1013) Allows an application to set the value of the data to be published under a given **DDS_Topic** (p. 175).

QoS:

DDS_DataWriterQos (p. 1666)

Status:

DDS_LIVELINESS_LOST_STATUS (p. 1231), **DDS_LivelinessLostStatus** (p. 1822);
DDS_OFFERED_DEADLINE_MISSED_STATUS (p. 1228), **DDS_OfferedDeadlineMissedStatus** (p. 1861);
DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 1229), **DDS_OfferedIncompatibleQosStatus** (p. 1862);
DDS_PUBLICATION_MATCHED_STATUS (p. 1232), **DDS_PublicationMatchedStatus** (p. 1914);
DDS_RELIABLE_READER_ACTIVITY_CHANGED_STATUS (p. 1234), **DDS_ReliableReaderActivity**↔
ChangedStatus (p. 1940);
DDS_RELIABLE_WRITER_CACHE_CHANGED_STATUS (p. 1234), **DDS_ReliableWriterCache**↔
Status (p. 1942).

Listener:

DDS_DataWriterListener (p. 1644)

A **DDS_DataWriter** (p. 485) is attached to exactly one **DDS_Publisher** (p. 443), that acts as a factory for it.

A **DDS_DataWriter** (p. 485) is bound to exactly one **DDS_Topic** (p. 175) and therefore to exactly one data type. The **DDS_Topic** (p. 175) must exist prior to the **DDS_DataWriter** (p. 485)'s creation.

DDS_DataWriter (p. 485) is an abstract class. It must be specialized for each particular application data-type (see **USER_DATA** (p. 1343)). The additional methods or functions that must be defined in the auto-generated class for a hypothetical application type **Foo** (p. 2109) are specified in the example type **DDS_DataWriter** (p. 485).

The following operations may be called even if the **DDS_DataWriter** (p. 485) is not enabled. Other operations will fail with **DDS_RETCODE_NOT_ENABLED** (p. 1222) if called on a disabled **DDS_DataWriter** (p. 485):

- The base-class operations **DDS_DataWriter_set_qos** (p. 550), **DDS_DataWriter_get_qos** (p. 552), **DDS_DataWriter_set_listener** (p. 553), **DDS_DataWriter_get_listener** (p. 554), **DDS_Entity_enable** (p. 1363), **DDS_Entity_get_statuscondition** (p. 1365) and **DDS_Entity_get_status_changes** (p. 1365)
- **DDS_DataWriter_get_liveliness_lost_status** (p. 545), **DDS_DataWriter_get_offered_deadline_missed_status** (p. 546), **DDS_DataWriter_get_offered_incompatible_qos_status** (p. 546), **DDS_DataWriter_get_publication_matched_status** (p. 546), **DDS_DataWriter_get_reliable_writer_cache_changed_status** (p. 547), **DDS_DataWriter_get_reliable_reader_activity_changed_status** (p. 547) **DDS_DataWriter_get_service_request_accepted_status** (p. 550)

Several **DDS_DataWriter** (p. 485) may operate in different threads. If they share the same **DDS_Publisher** (p. 443), the middleware guarantees that its operations are thread-safe.

See also

FooDataWriter (p. 2113)

Operations Allowed in Listener Callbacks (p. 1818)

5.17.3.2 DDS_DataWriterListener_OfferedDeadlineMissedCallback

```
typedef void(* DDS_DataWriterListener_OfferedDeadlineMissedCallback) (void *listener_data, DDS_DataWriter *writer, const struct DDS_OfferedDeadlineMissedStatus *status)
```

Prototype of a **DDS_DataWriterListener** (p. 1644) on `_offered_deadline_missed` function.

5.17.3.3 DDS_DataWriterListener_LivelinessLostCallback

```
typedef void(* DDS_DataWriterListener_LivelinessLostCallback) (void *listener_data, DDS_DataWriter *writer, const struct DDS_LivelinessLostStatus *status)
```

Prototype of a **DDS_DataWriterListener** (p. 1644) on `_liveliness_lost` function.

Parameters

<i>listener_data</i>	<< out >> (p. 1013) Data associated with the listener when the listener is set
<i>writer</i>	<< out >> (p. 1013) Locally created DDS_DataWriter (p. 485) that triggers the listener callback
<i>status</i>	<< out >> (p. 1013) Current liveness lost status of the locally created DDS_DataWriter (p. 485)

5.17.3.4 DDS_DataWriterListener_OfferedIncompatibleQosCallback

```
typedef void(* DDS_DataWriterListener_OfferedIncompatibleQosCallback) (void *listener_data, DDS_↵
_DataWriter *writer, const struct DDS_OfferedIncompatibleQosStatus *status)
```

Prototype of a **DDS_DataWriterListener** (p. 1644) on_offered_incompatible_qos function.

Parameters

<i>listener_data</i>	<< out >> (p. 1013) Data associated with the listener when the listener is set
<i>writer</i>	<< out >> (p. 1013) Locally created DDS_DataWriter (p. 485) that triggers the listener callback
<i>status</i>	<< out >> (p. 1013) Current offered incompatible qos status of the locally created DDS_DataWriter (p. 485)

5.17.3.5 DDS_DataWriterListener_PublicationMatchedException

```
typedef void(* DDS_DataWriterListener_PublicationMatchedException) (void *listener_data, DDS_↵
DataWriter *writer, const struct DDS_PublicationMatchedException *status)
```

Prototype of a **DDS_DataWriterListener** (p. 1644) on_publication_matched function.

5.17.3.6 DDS_DataWriterListener_ReliableWriterCacheChangedCallback

```
typedef void(* DDS_DataWriterListener_ReliableWriterCacheChangedCallback) (void *listener_data,
DDS_DataWriter *writer, const struct DDS_ReliableWriterCacheChangedStatus *status)
```

<<**extension**>> (p. 1012) A change has occurred in the writer's cache.

This callback is triggered when the cache becomes empty or full, or when the number of unacknowledged samples reaches a high or low watermark.

Parameters

<i>listener_data</i>	<< out >> (p. 1013) Data associated with the listener when the listener is set
<i>writer</i>	<< out >> (p. 1013) Locally created DDS_DataWriter (p. 485) that triggers the listener callback
<i>status</i>	<< out >> (p. 1013) Current reliable writer cache changed status of the locally created DDS_DataWriter (p. 485)

5.17.3.7 DDS_DataWriterListener_ReliableReaderActivityChangedCallback

```
typedef void(* DDS_DataWriterListener_ReliableReaderActivityChangedCallback) (void *listener_data,
DDS_DataWriter *writer, const struct DDS_ReliableReaderActivityChangedStatus *status)
```

<<**extension**>> (p. 1012) A matched reliable reader has become active or become inactive.

Parameters

<i>listener_data</i>	<< out >> (p. 1013) Data associated with the listener when the listener is set
<i>writer</i>	<< out >> (p. 1013) Locally created DDS_DataWriter (p. 485) that triggers the listener callback
<i>status</i>	<< out >> (p. 1013) Current reliable reader activity changed status of the locally created DDS_DataWriter (p. 485)

5.17.3.8 DDS_DataWriterListener_SampleRemovedCallback

```
typedef void(* DDS_DataWriterListener_SampleRemovedCallback) (void *listener_data, DDS_DataWriter
*writer, const struct DDS_Cookie_t *cookie)
```

<<**extension**>> (p. 1012) A sample has been removed from the DataWriter's queue.

Parameters

<i>listener_data</i>	<< out >> (p. 1013) Data associated with the listener when the listener is set
<i>writer</i>	<< out >> (p. 1013) Locally created DDS_DataWriter (p. 485) that triggers the listener callback
<i>cookie</i>	<< out >> (p. 1013) • If this sample was written with FooDataWriter_write_w_params (p. 501), this field contains a copy of the cookie set in DDS_WriteParams_t (p. 2101). • If this writer uses Zero Copy Transfer Over Shared Memory (p. 210) "Zero Copy transfer over shared memory", this field contains the absolute address of the sample that is removed. The address of the sample can be obtained by using DDS_Cookie_to_pointer (p. 1386).

See also

FooDataWriter_get_loan (p. 508)

5.17.3.9 DDS_DataWriterListener_InstanceReplacedCallback

```
typedef void(* DDS_DataWriterListener_InstanceReplacedCallback) (void *listener_data,  DDS_DataWriter *writer, const  DDS_InstanceHandle_t *handle)
```

<<**extension**>> (p. 1012) An instance previously registered with the writer has been replaced

Parameters

<i>listener_data</i>	<< out >> (p. 1013) Data associated with the listener when the listener is set
<i>writer</i>	<< out >> (p. 1013) Locally created DDS_DataWriter (p. 485) that triggers the listener callback
<i>handle</i>	<< out >> (p. 1013) Handle of the replaced instance

5.17.3.10 DDS_DataWriterListener_OnApplicationAcknowledgmentCallback

```
typedef void(* DDS_DataWriterListener_OnApplicationAcknowledgmentCallback) (void *listener_data, DDS_DataWriter *writer, const struct  DDS_AcknowledgmentInfo *info)
```

<<**extension**>> (p. 1012) An application-level acknowledgment has been received for a sample

Parameters

<i>listener_data</i>	<< out >> (p. 1013) Data associated with the listener when the listener is set
<i>writer</i>	<< out >> (p. 1013) Locally created DDS_DataWriter (p. 485) that triggers the listener callback
<i>info</i>	<< out >> (p. 1013) DDS_AcknowledgmentInfo (p. 1535) of the acknowledged sample

5.17.3.11 DDS_DataWriterListener_ServiceRequestAcceptedCallback

```
typedef void(* DDS_DataWriterListener_ServiceRequestAcceptedCallback) (void *listener_data,  DDS_DataWriter *writer, const struct  DDS_ServiceRequestAcceptedStatus *status)
```

<<**extension**>> (p. 1012) a **DDS_ServiceRequest** (p. 2001) for the **DDS_TopicQuery** (p. 707) service is dispatched to this **DDS_DataWriter** (p. 485).

Parameters

<i>listener_data</i>	<< out >> (p. 1013) Data associated with the listener when the listener is set
<i>writer</i>	<< out >> (p. 1013) Locally created DDS_DataWriter (p. 485) that triggers the listener callback
<i>status</i>	<< out >> (p. 1013) Current service request accepted status of locally created DDS_DataWriter (p. 485)

See also

Topic Queries (p. 703)

5.17.4 Function Documentation

5.17.4.1 FooDataWriter_narrow()

```
FooDataWriter * FooDataWriter_narrow (
    DDS_DataWriter * writer )
```

Narrow the given **DDS_DataWriter** (p. 485) pointer to a **FooDataWriter** (p. 2113) pointer.

Check if the given `writer` is of type **FooDataWriter** (p. 2113).

Parameters

<i>writer</i>	<< in >> (p. 1013) Base-class DDS_DataWriter (p. 485) to be converted to the auto-generated class FooDataWriter (p. 2113) that extends DDS_DataWriter (p. 485). Cannot be NULL.

Returns

FooDataWriter (p. 2113) if `writer` is of type **Foo** (p. 2109). Return NULL otherwise.

5.17.4.2 FooDataWriter_as_datawriter()

```
DDS_DataWriter * FooDataWriter_as_datawriter (
    FooDataWriter * writer )
```

Widen the given **FooDataWriter** (p. 2113) pointer to a **DDS_DataWriter** (p. 485) pointer.

Parameters

<i>writer</i>	<< <i>in</i> >> (p. 1013) auto-generated FooDataWriter (p. 2113) to be converted to the Base-class DDS_DataWriter (p. 485). Cannot be NULL.

Returns

DDS_DataWriter (p. 485).

5.17.4.3 FooDataWriter_register_instance()

```
DDS_InstanceHandle_t FooDataWriter_register_instance (
    FooDataWriter * self,
    const Foo * instance_data )
```

Informs RTI Connexx 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. 228). 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()` or `dispose()` operations.

The operation gives RTI Connexx 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. 228) as the **DDS_InstanceHandle_t** (p. 214) parameter to the `write` or `dispose` operation and RTI Connexx 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. 496), **FooDataWriter_write_w_timestamp** (p. 500), **FooDataWriter_dispose** (p. 502) and **FooDataWriter_dispose_w_timestamp** (p. 503) and the handle used in conjunction with the data for those calls.

When this operation is used, RTI Connexx will automatically supply the value of the `source_timestamp` that is used.

This operation may fail and return **DDS_HANDLE_NIL** (p. 228) if **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1951) 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 **DDS_DataWriter** (p. 485) has been enabled. Otherwise, **DDS_HANDLE_NIL** (p. 228) will be returned.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>instance_data</i>	<< <i>in</i> >> (p. 1013) The instance that should be registered. Of this instance, only the fields that represent the key are examined by the function. Cannot be NULL.

Returns

For keyed data type, a handle that can be used in the calls that take a **DDS_InstanceHandle_t** (p. 214), such as `write`, `dispose`, `unregister_instance`, or return **DDS_HANDLE_NIL** (p. 228) on failure. If the `instance_data` is of a data type that has no keys, this function always returns **DDS_HANDLE_NIL** (p. 228).

See also

FooDataWriter_unregister_instance (p. 493), **FooDataWriter_get_key_value** (p. 505), **Relationship between registration, liveliness and ownership** (p. 1866)

5.17.4.4 FooDataWriter_register_instance_w_timestamp()

```
DDS_InstanceHandle_t FooDataWriter_register_instance_w_timestamp (
    FooDataWriter * self,
    const Foo * instance_data,
    const struct DDS_Time_t * source_timestamp )
```

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. 1273) QoS policy for details.

This operation may fail and return **DDS_HANDLE_NIL** (p. 228) if **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1951) limit has been exceeded.

This operation can only be called after **DDS_DataWriter** (p. 485) has been enabled. Otherwise, **DDS_HANDLE_NIL** (p. 228) will be returned.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>instance_data</i>	<< <i>in</i> >> (p. 1013) The instance that should be registered. Of this instance, only the fields that represent the key are examined by the function. Cannot be NULL.
<i>source_timestamp</i>	<< <i>in</i> >> (p. 1013) The timestamp value must be greater than or equal to the timestamp value used in the last writer operation (used in a <i>register</i> , <i>unregister</i> , <i>dispose</i> , or <i>write</i> , 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. Cannot be NULL.

Returns

For keyed data type, return a handle that can be used in the calls that take a **DDS_InstanceHandle_t** (p. 214), such as `write`, `dispose`, `unregister_instance`, or return **DDS_HANDLE_NIL** (p. 228) on failure. If the `instance_data` is of a data type that has no keys, this function always return **DDS_HANDLE_NIL** (p. 228).

See also

FooDataWriter_unregister_instance (p. 493), **FooDataWriter_get_key_value** (p. 505)

5.17.4.5 FooDataWriter_register_instance_w_params()

```
DDS_InstanceHandle_t FooDataWriter_register_instance_w_params (
    FooDataWriter * self,
    const Foo * instance_data,
    struct DDS_WriteParams_t * params )
```

Performs the same function as **FooDataWriter_register_instance** (p. 491) and **FooDataWriter_register_instance_w_timestamp** (p. 492) except that it also provides the values contained in `params`.

See also

FooDataWriter_write_w_params (p. 501)

5.17.4.6 FooDataWriter_unregister_instance()

```
DDS_ReturnCode_t FooDataWriter_unregister_instance (
    FooDataWriter * self,
    const Foo * instance_data,
    const DDS_InstanceHandle_t * handle )
```

Reverses the action of **FooDataWriter_register_instance** (p. 491).

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. 491). Otherwise, this operation may fail with **DDS_RETCODE_BAD_PARAMETER** (p. 1222).

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 Connexx will automatically supply the value of the `source_timestamp` that is used.

This operation informs RTI Connexx that the **DDS_DataWriter** (p. 485) is no longer going to provide any information about the instance. This operation also indicates that RTI Connexx 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. 228) 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. 228), 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. 1222).

RTI Connext will not detect the error when the `handle` is any value other than **DDS_HANDLE_NIL** (p. 228), 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()` operation is for the instance as indicated by the `handle`.

If, after a **FooDataWriter_unregister_instance** (p. 493), the application wants to modify (**FooDataWriter_write** (p. 496) or **FooDataWriter_dispose** (p. 502)) an instance, it has to register it again, or else use the special `handle` value **DDS_HANDLE_NIL** (p. 228).

This operation does not indicate that the instance is deleted (that is the purpose of **FooDataWriter_dispose** (p. 502)). The operation **FooDataWriter_unregister_instance** (p. 493) just indicates that the **DDS_DataWriter** (p. 485) no longer has anything to say about the instance. **DDS_DataReader** (p. 615) entities that are reading the instance may receive a sample with **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 715) for the instance, unless there are other **DDS_DataWriter** (p. 485) objects writing that same instance.

DDS_WriterDataLifecycleQosPolicy::autodispose_unregistered_instances (p. 2107) controls whether instances are automatically disposed when they are unregistered.

This operation can affect the ownership of the data instance (see **OWNERSHIP** (p. 1301)). If the **DDS_DataWriter** (p. 485) was the exclusive owner of the instance, then calling `unregister_instance()` will relinquish that ownership.

If **DDS_ReliabilityQosPolicy::kind** (p. 1939) is set to **DDS_RELIABLE_RELIABILITY_QOS** (p. 1323) 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. 1939); if this writer is still unable to unregister after that period, this function will fail with **DDS_RETCODE_TIMEOUT** (p. 1222).

Parameters

<i>instance_data</i>	<< <i>in</i> >> (p. 1013) The instance that should be unregistered. If Foo (p. 2109) has a key and <code>instance_handle</code> is DDS_HANDLE_NIL (p. 228), only the fields that represent the key are examined by the function. Otherwise, <code>instance_data</code> is not used. If <code>instance_data</code> is used, it must represent an instance that has been registered. Otherwise, this function may fail with DDS_RETCODE_BAD_PARAMETER (p. 1222). If Foo (p. 2109) has a key, <code>instance_data</code> can be NULL only if <code>handle</code> is not DDS_HANDLE_NIL (p. 228). Otherwise, this function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222).
<i>handle</i>	<< <i>in</i> >> (p. 1013) represents the instance to be unregistered. If Foo (p. 2109) has a key and <code>handle</code> is DDS_HANDLE_NIL (p. 228), <code>handle</code> is not used and <code>instance</code> is deduced from <code>instance_data</code> . If Foo (p. 2109) has no key, <code>handle</code> is not used. If <code>handle</code> is used, it must represent an instance that has been registered. Otherwise, this function may fail with DDS_RETCODE_BAD_PARAMETER (p. 1222). This function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if <code>handle</code> is NULL. If Foo (p. 2109) has a key, <code>handle</code> cannot be DDS_HANDLE_NIL (p. 228) if <code>instance_data</code> is NULL. Otherwise, this function will report the error DDS_RETCODE_BAD_PARAMETER (p. 1222).
<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_TIMEOUT** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222)

See also

FooDataWriter_register_instance (p. 491)

FooDataWriter_unregister_instance_w_timestamp (p. 495)

FooDataWriter_get_key_value (p. 505)

Relationship between registration, liveliness and ownership (p. 1866)

5.17.4.7 FooDataWriter_unregister_instance_w_timestamp()

```
DDS_ReturnCode_t FooDataWriter_unregister_instance_w_timestamp (
    FooDataWriter * self,
    const Foo * instance_data,
    const DDS_InstanceHandle_t * handle,
    const struct DDS_Time_t * source_timestamp )
```

Performs the same function as **FooDataWriter_unregister_instance** (p. 493) 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. 1273) 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. 493) operation.

This operation may block and may time out (**DDS_RETCODE_TIMEOUT** (p. 1222)) under the same circumstances described for the `unregister_instance` operation.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>instance_data</i>	<< <i>in</i> >> (p. 1013) The instance that should be unregistered. If Foo (p. 2109) <i>has</i> a key and <code>instance_handle</code> is DDS_HANDLE_NIL (p. 228), only the fields that represent the key are examined by the function. Otherwise, <code>instance_data</code> is not used. If <code>instance_data</code> is used, it must represent an instance that has been registered. Otherwise, this function may fail with DDS_RETCODE_BAD_PARAMETER (p. 1222). If Foo (p. 2109) <i>has</i> a key, <code>instance_data</code> can be NULL only if <code>handle</code> is not DDS_HANDLE_NIL (p. 228). Otherwise, this function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222).
<i>handle</i>	<< <i>in</i> >> (p. 1013) represents the <code>instance</code> to be unregistered. If Foo (p. 2109) <i>has</i> a key and <code>handle</code> is DDS_HANDLE_NIL (p. 228), <code>handle</code> is not used and <code>instance</code> is deduced from <code>instance_data</code> . If Foo (p. 2109) <i>has</i> no key, <code>handle</code> is not used. If <code>handle</code> is used, it must represent an instance that has been registered. Otherwise, this function may fail with DDS_RETCODE_BAD_PARAMETER (p. 1222). This function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if <code>handle</code> is NULL. If Foo (p. 2109) <i>has</i> a key, <code>handle</code> cannot be DDS_HANDLE_NIL (p. 228) if <code>instance_data</code> is NULL. Otherwise, this function will report the error DDS_RETCODE_BAD_PARAMETER (p. 1222).
<i>source_timestamp</i>	<< <i>in</i> >> (p. 1013) The timestamp value must be greater than or equal to the timestamp value used in the last writer operation (used in a <i>register</i> , <i>unregister</i> , <i>dispose</i> , or <i>write</i> , 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. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_TIMEOUT** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

FooDataWriter_register_instance (p. 491)

FooDataWriter_unregister_instance (p. 493)

FooDataWriter_get_key_value (p. 505)

5.17.4.8 FooDataWriter_unregister_instance_w_params()

```
DDS_ReturnCode_t FooDataWriter_unregister_instance_w_params (
    FooDataWriter * self,
    const Foo * instance_data,
    struct DDS_WriteParams_t * params )
```

Performs the same function as **FooDataWriter_unregister_instance** (p. 493) and **FooDataWriter_unregister_instance_w_timestamp** (p. 495) except that it also provides the values contained in *params*.

See also

FooDataWriter_write_w_params (p. 501)

FooDataWriter_dispose_w_params (p. 504)

5.17.4.9 FooDataWriter_write()

```
DDS_ReturnCode_t FooDataWriter_write (
    FooDataWriter * self,
    const Foo * instance_data,
    const DDS_InstanceHandle_t * handle )
```

Modifies the value of a data instance.

When this operation is used, RTI Connex will automatically supply the value of the *source_timestamp* that is made available to **DDS_DataReader** (p. 615) objects by means of the *source_timestamp* attribute inside the **DDS_SampleInfo** (p. 1985). (Refer to **DDS_SampleInfo** (p. 1985) and **DESTINATION_ORDER** (p. 1273) QoS policy for details).

As a side effect, this operation asserts liveness on the **DDS_DataWriter** (p. 485) itself, the **DDS_Publisher** (p. 443) and the **DDS_DomainParticipant** (p. 77).

Note that the special value **DDS_HANDLE_NIL** (p. 228) 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. 228), 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. 1222).

RTI Connex will not detect the error when the `handle` is any value other than **DDS_HANDLE_NIL** (p. 228), 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 Connex will treat as if the `write()` operation is for the instance as indicated by the `handle`.

This operation may block if the **RELIABILITY** (p. 1321) `kind` is set to **DDS_RELIABLE_RELIABILITY_QOS** (p. 1323) and the modification would cause data to be lost or else cause one of the limits specified in the **RESOURCE_LIMITS** (p. 1325) to be exceeded.

This operation will not block when using **DDS_BEST_EFFORT_RELIABILITY_QOS** (p. 1323). If you are using **BEST_EFFORT** Reliability in combination with **DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 1319), 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. 1805).

If **DDS_ReliabilityQosPolicy::max_blocking_time** (p. 1939) elapses before the **DDS_DataWriter** (p. 485) can store the modification without exceeding the limits, the operation will fail and return **DDS_RETCODE_TIMEOUT** (p. 1222) for **KEEP_ALL** configurations.

Here is how the write operation behaves when **DDS_KEEP_LAST_HISTORY_QOS** (p. 1293) and **DDS_RELIABLE_RELIABILITY_QOS** (p. 1323) are used:

- The send window size is determined by the **DDS_RtpsReliableWriterProtocol_t::max_send_window_size** (p. 1973) and **DDS_RtpsReliableWriterProtocol_t::min_send_window_size** (p. 1973) fields in the **DDS_DataWriterProtocolQosPolicy** (p. 1649). 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. 1939) expires.
- Then, the **DDS_DataWriter** (p. 485) 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. 1292)) 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. 1222). 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_ResourceLimitsQosPolicy::max_instances** (p. 1951) is exceeded, the **DataWriter** will try to replace an existing instance (and its samples) according to the value of **DDS_DataWriterResourceLimitsQosPolicy::instance_replacement** (p. 1677) (see **DDS_DataWriterResourceLimitsInstanceReplacementKind** (p. 1269)).
 - If no instance can be replaced, the write operation returns **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).
- If **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1951) 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. 1222).

Here is how the write operation behaves when **DDS_KEEP_ALL_HISTORY_QOS** (p. 1293) and **DDS_RELIABLE_↵RELIABILITY_QOS** (p. 1323) are used:

- The send window size is determined by the **DDS_RtpsReliableWriterProtocol_t::max_send_window_size** (p. 1973) and **DDS_RtpsReliableWriterProtocol_t::min_send_window_size** (p. 1973) fields in the **DATA_↵WRITER_PROTOCOL** (p. 1268). 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. 1939) expires.
 - If the `max_blocking_time` expires, the write operation returns **DDS_RETCODE_TIMEOUT** (p. 1222).
- 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. 1677) (see **DDS_DataWriter_↵ResourceLimitsInstanceReplacementKind** (p. 1269)).
 - * If no instance can be replaced, the write operation returns **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).
 - If **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1951) 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. 1939) expires. If the `max_blocking_↵_time` expires, the write operation will return **DDS_RETCODE_TIMEOUT** (p. 1222).
 - If **DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1952) 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. 1222).
 - The sample is added to the writer history, and the write operation returns **DDS_RETCODE_OK** (p. 1222).

If there are no instance resources left, this operation may fail with **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222). Calling **FooDataWriter_unregister_instance** (p. 493) may help freeing up some resources.

This operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) 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. 508) is modified with this operation, then all instances modified thereafter should be from **FooDataWriter_get_loan** (p. 508).

See **DDS_ReliabilityQosPolicyAcknowledgmentModeKind** (p. 1323) 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.1939) set to **DDS_APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE** (p.1324) or **DDS_APPLICATION_AUTO_ACKNOWLEDGMENT_MODE** (p.1323). 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.1547)), 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.1951) or **DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p.1952) 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.1939).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>instance_data</i>	<< <i>in</i> >> (p. 1013) The data to write.

This function will fail with **DDS_RETCODE_BAD_PARAMETER** (p.1222) if *instance_data* is NULL.

Parameters

<i>handle</i>	<< <i>in</i> >> (p. 1013) Either the handle returned by a previous call to FooDataWriter_register_instance (p.491), or else the special value DDS_HANDLE_NIL (p.228). If Foo (p.2109) has a key and <i>handle</i> is not DDS_HANDLE_NIL (p.228), <i>handle</i> must represent a registered instance of type Foo (p.2109). Otherwise, this function may fail with DDS_RETCODE_BAD_PARAMETER (p.1222). This function will fail with DDS_RETCODE_BAD_PARAMETER (p.1222) if <i>handle</i> is NULL.

Returns

One of the **Standard Return Codes** (p.1221), **DDS_RETCODE_TIMEOUT** (p.1222), **DDS_RETCODE_PRECONDITION_NOT_MET** (p.1222), **DDS_RETCODE_OUT_OF_RESOURCES** (p.1222), or **DDS_RETCODE_NOT_ENABLED** (p.1222).

MT Safety:

It is UNSAFE to modify `instance_data` before the operation has finished. The operation is otherwise SAFE.

See also

DDS_DataReader (p. 615)

FooDataWriter_write_w_timestamp (p. 500)

DESTINATION_ORDER (p. 1273)

5.17.4.10 FooDataWriter_write_w_timestamp()

```
DDS_ReturnCode_t FooDataWriter_write_w_timestamp (
    FooDataWriter * self,
    const Foo * instance_data,
    const DDS_InstanceHandle_t * handle,
    const struct DDS_Time_t * source_timestamp )
```

Performs the same function as **FooDataWriter_write** (p. 496) except that it also provides the value for the `source_timestamp`.

Explicitly provides the timestamp that will be available to the **DDS_DataReader** (p. 615) objects by means of the `source_timestamp` attribute inside the **DDS_SampleInfo** (p. 1985). (Refer to **DDS_SampleInfo** (p. 1985) and **DESTINATION_ORDER** (p. 1273) 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. 496) operation.

This operation may block and time out (**DDS_RETCODE_TIMEOUT** (p. 1222)) under the same circumstances described for **FooDataWriter_write** (p. 496).

If there are no instance resources left, this operation may fail with **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222). Calling **FooDataWriter_unregister_instance** (p. 493) may help free up some resources.

This operation may fail with **DDS_RETCODE_BAD_PARAMETER** (p. 1222) under the same circumstances described for the write operation.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>instance_data</i>	<< <i>in</i> >> (p. 1013) The data to write. This function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if <code>instance_data</code> is NULL.
<i>handle</i>	<< <i>in</i> >> (p. 1013) Either the handle returned by a previous call to FooDataWriter_register_instance (p. 491), or else the special value DDS_HANDLE NIL (p. 228). If Foo (p. 2109) has a key and <code>handle</code> is not DDS_HANDLE NIL (p. 228), <code>handle</code> must represent a registered instance of type Foo (p. 2109). Otherwise, this function may fail with DDS_RETCODE_BAD_PARAMETER (p. 1222). This function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if <code>handle</code> is NULL.

Parameters

<code>source_timestamp</code>	<<<i>in</i>>> (p. 1013) When using DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS (p. 1274) the timestamp value must be greater than or equal to the timestamp value used in the last writer operation (<i>register</i>, <i>unregister</i>, <i>dispose</i>, or <i>write</i>, 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. 1688), 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. 1688), the function will return DDS_RETCODE_BAD_PARAMETER (p. 1222).

Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_TIMEOUT** (p. 1222), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222), or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

FooDataWriter_write (p. 496)
DDS_DataReader (p. 615)
DESTINATION_ORDER (p. 1273)

5.17.4.11 FooDataWriter_write_w_params()

```
DDS_ReturnCode_t FooDataWriter_write_w_params (
    FooDataWriter * self,
    const Foo * instance_data,
    struct DDS_WriteParams_t * params )
```

Performs the same function as **FooDataWriter_write** (p. 496) and **FooDataWriter_write_w_timestamp** (p. 500) 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. 1222)) under the same circumstances described for **FooDataWriter_write** (p. 496).

If there are no instance resources left, this operation may fail with **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222). Calling **FooDataWriter_unregister_instance_w_params** (p. 496) may help free up some resources.

This operation may fail with **DDS_RETCODE_BAD_PARAMETER** (p. 1222) under the same circumstances described for the **FooDataWriter_write** (p. 496).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>instance_data</i>	<< <i>in</i> >> (p. 1013) The data to write. This function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if <i>instance_data</i> is NULL.
<i>params</i>	<< <i>inout</i> >> (p. 1013) The write parameters. Note that this is an inout parameter if you activate DDS_WriteParams_t::replace_auto (p. 2102); otherwise it won't be modified. This function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if <i>params</i> is NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_TIMEOUT** (p. 1222), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

FooDataWriter_write (p. 496)

DDS_DataReader (p. 615)

5.17.4.12 FooDataWriter_dispose()

```
DDS_ReturnCode_t FooDataWriter_dispose (
    FooDataWriter * self,
    const Foo * instance_data,
    const DDS_InstanceHandle_t * instance_handle )
```

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 **DDS_DataWriter** (p. 485) communicates this state change to **DDS_DataReader** (p. 615) 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. 1989). Disposed instances have the value **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 715).

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_WriterDataLifecycleQosPolicy::autopurge_disposed_instances_delay** (p. 2108).

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. 1932).

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 Connex will automatically supply the value of the *source_timestamp* that is made available to **DDS_DataReader** (p. 615) objects by means of the *source_timestamp* attribute inside the **DDS_SampleInfo** (p. 1985).

The constraints on the values of the handle parameter and the corresponding error behavior are the same specified for the **FooDataWriter_unregister_instance** (p. 493) operation.

The special value **DDS_HANDLE_NIL** (p. 228) 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. 228), 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. 1222).

RTI Connext will not detect the error when the *instance_handle* is any value other than **DDS_HANDLE_NIL** (p. 228), 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. 1222)) under the same circumstances described for **FooDataWriter_write** (p. 496).

If there are no instance resources left, this operation may fail with **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222). Calling **FooDataWriter_unregister_instance** (p. 493) may help free up some resources.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>instance_data</i>	<< <i>in</i> >> (p. 1013) The data to dispose. If Foo (p. 2109) has a key and <i>instance_handle</i> is DDS_HANDLE_NIL (p. 228), only the fields that represent the key are examined by the function. Otherwise, <i>instance_data</i> is not used. If Foo (p. 2109) has a key, <i>instance_data</i> can be NULL only if <i>instance_handle</i> is not DDS_HANDLE_NIL (p. 228). Otherwise, this function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222).
<i>instance_handle</i>	<< <i>in</i> >> (p. 1013) Either the handle returned by a previous call to FooDataWriter_register_instance (p. 491), or else the special value DDS_HANDLE_NIL (p. 228). If Foo (p. 2109) has a key and <i>instance_handle</i> is DDS_HANDLE_NIL (p. 228), <i>instance_handle</i> is not used and it is deduced from <i>instance_data</i> . If Foo (p. 2109) has no key, <i>instance_handle</i> is not used. If <i>instance_handle</i> is used, it must represent an instance of type Foo (p. 2109) that has been written or registered with this writer. Otherwise, this function fail with DDS_RETCODE_BAD_PARAMETER (p. 1222). This function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if <i>instance_handle</i> is NULL. If Foo (p. 2109) has a key, <i>instance_handle</i> cannot be DDS_HANDLE_NIL (p. 228) if <i>instance_data</i> is NULL. Otherwise, this function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222).

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_TIMEOUT** (p. 1222), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

FooDataWriter_dispose_w_timestamp (p. 503)

Relationship between registration, liveliness and ownership (p. 1866)

5.17.4.13 FooDataWriter_dispose_w_timestamp()

```
DDS_ReturnCode_t FooDataWriter_dispose_w_timestamp (
    FooDataWriter * self,
    const Foo * instance_data,
    const DDS_InstanceHandle_t * instance_handle,
    const struct DDS_Time_t * source_timestamp )
```

Performs the same functions as `dispose` except that the application provides the value for the `source_timestamp` that is made available to **DDS_DataReader** (p. 615) objects by means of the `source_timestamp` attribute inside the **DDS_SampleInfo** (p. 1985).

The constraints on the values of the `handle` parameter and the corresponding error behavior are the same specified for the **FooDataWriter_dispose** (p. 502) operation.

This operation may block and time out (**DDS_RETCODE_TIMEOUT** (p. 1222)) under the same circumstances described for **FooDataWriter_write** (p. 496).

If there are no instance resources left, this operation may fail with **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222). Calling **FooDataWriter_unregister_instance** (p. 493) may help freeing up some resources.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>instance_data</i>	<< <i>in</i> >> (p. 1013) The data to dispose. If Foo (p. 2109) has a key and <code>instance_handle</code> is DDS_HANDLE_NIL (p. 228), only the fields that represent the key are examined by the function. Otherwise, <code>instance_data</code> is not used. If Foo (p. 2109) has a key, <code>instance_data</code> can be NULL only if <code>instance_handle</code> is not DDS_HANDLE_NIL (p. 228). Otherwise, this function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222).
<i>instance_handle</i>	<< <i>in</i> >> (p. 1013) Either the handle returned by a previous call to FooDataWriter_register_instance (p. 491), or else the special value DDS_HANDLE_NIL (p. 228). If Foo (p. 2109) has a key and <code>handle</code> is not DDS_HANDLE_NIL (p. 228), <code>handle</code> must represent a registered instance of type Foo (p. 2109). Otherwise, this function may fail with DDS_RETCODE_BAD_PARAMETER (p. 1222). This function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if <code>handle</code> is NULL.
<i>source_timestamp</i>	<< <i>in</i> >> (p. 1013) The timestamp value must be greater than or equal to the timestamp value used in the last writer operation (used in a <i>register</i> , <i>unregister</i> , <i>dispose</i> , or <i>write</i> , 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 DDS_DataReader (p. 615) objects by means of the <code>source_timestamp</code> attribute inside the DDS_SampleInfo (p. 1985). Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_TIMEOUT** (p. 1222), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

FooDataWriter_dispose (p. 502)

5.17.4.14 FooDataWriter_dispose_w_params()

```
DDS_ReturnCode_t FooDataWriter_dispose_w_params (
    FooDataWriter * self,
    const Foo * instance_data,
    struct DDS_WriteParams_t * params )
```

Performs the same function as **FooDataWriter_dispose** (p. 502) and **FooDataWriter_dispose_w_timestamp** (p. 503) except that it also provides the values contained in `params`.

See also

FooDataWriter_write_w_params (p. 501)

5.17.4.15 FooDataWriter_get_key_value()

```
DDS_ReturnCode_t FooDataWriter_get_key_value (
    FooDataWriter * self,
    Foo * key_holder,
    const DDS_InstanceHandle_t * handle )
```

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 function has no effect and exits with no error.

For keyed data types, this operation may fail with **DDS_RETCODE_BAD_PARAMETER** (p. 1222) if the `handle` does not correspond to an existing data-object known to the **DDS_DataWriter** (p. 485).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>key_holder</i>	<< <i>inout</i> >> (p. 1013) a user data type specific key holder, whose <code>key</code> fields are filled by this operation. If Foo (p. 2109) has no key, this function has no effect.

This function will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 1222) if `key_holder` is NULL.

Parameters

<i>handle</i>	<< <i>in</i> >> (p. 1013) the <code>instance</code> whose key is to be retrieved. If Foo (p. 2109) has a key, <code>handle</code> must represent a registered instance of type Foo (p. 2109). Otherwise, this function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222). If Foo (p. 2109) has a key and <code>handle</code> is DDS_HANDLE_NIL (p. 228), this function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222). This function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if <code>handle</code> is NULL.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

FooDataReader_get_key_value (p. 647)

5.17.4.16 FooDataWriter_lookup_instance()

```
DDS_InstanceHandle_t FooDataWriter_lookup_instance (
    FooDataWriter * self,
    const Foo * key_holder )
```

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 Connex is unable to provide an instance handle, RTI Connex will return the special value `HANDLE_NIL`.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>key_holder</i>	<< <i>in</i> >> (p. 1013) a user data type specific key holder.

Returns

the instance handle associated with this instance. If **Foo** (p. 2109) has no key, this function has no effect and returns **DDS_HANDLE_NIL** (p. 228)

5.17.4.17 FooDataWriter_create_data()

```
void * FooDataWriter_create_data (
    FooDataWriter * self )
```

Creates a data sample and initializes it.

The behavior of this API is identical to **FooTypeSupport_create_data** (p. 215).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

Newly created data type, or NULL on failure.

See also

FooDataWriter_delete_data (p. 507)

5.17.4.18 FooDataWriter_create_data_w_params()

```
void * FooDataWriter_create_data_w_params (
    FooDataWriter * self,
    const struct DDS_TypeAllocationParams_t * alloc_params )
```

Creates a data sample and initializes it.

The behavior of this API is identical to **FooTypeSupport_create_data_w_params** (p. 216).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>alloc_params</i>	<< <i>in</i> >> (p. 1013) 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_w_params (p. 508)

5.17.4.19 FooDataWriter_delete_data()

```
DDS_Boolean FooDataWriter_delete_data (
    FooDataWriter * self,
    Foo * sample )
```

Destroys a user data type instance.

The behavior of this API is identical to **FooTypeSupport_delete_data** (p. 217).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>sample</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) on success.

See also

FooDataWriter_create_data (p. 506)

5.17.4.20 FooDataWriter_delete_data_w_params()

```
void * FooDataWriter_delete_data_w_params (
    FooDataWriter * self,
    Foo * sample,
    const struct DDS_TypeDeallocationParams_t * dealloc_params )
```

Destroys a user data type instance.

The behavior of this API is identical to **FooTypeSupport_delete_data_w_params** (p. 218).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>sample</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>dealloc_params</i>	<< <i>in</i> >> (p. 1013) Whether or not to destroy pointers and/or optional members.

Returns

DDS_BOOLEAN_TRUE (p. 1201) on success.

See also

FooDataWriter_create_data_w_params (p. 507)

5.17.4.21 FooDataWriter_get_loan()

```
DDS_ReturnCode_t FooDataWriter_get_loan (
    FooDataWriter * self,
    Foo ** sample )
```

Gets a sample managed by the DataWriter.

This operation is supported while using **Zero Copy transfer** (p. 210) over shared memory" or \ref RTIFlatDataModule "FlatData language binding".

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. 1681) to **DDS_BOOLEAN_TRUE** (p. 1201). The **DDS_DataWriterResourceLimitsQosPolicy::writer_loaned_sample_allocation** (p. 1681) settings can be used to configure the DataWriter-managed sample pool.

FooDataWriter_get_loan (p. 508) fails with **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222) if **DDS_AllocationSettings_t::max_count** (p. 1538) samples have been loaned, and none of those samples has been written with **FooDataWriter_write** (p. 496) or discarded via **FooDataWriter_discard_loan** (p. 510).

Samples returned from **FooDataWriter_get_loan** (p. 508) 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. 508) 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. 508) 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. 508) 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. 510). 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. 1222) if an unmanaged sample was written with this DataWriter earlier. Similarly, **FooDataWriter_write** (p. 496) will fail to write an unmanaged sample if get_loan was called.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>sample</i>	<< <i>inout</i> >> (p. 1013) address of a user data type pointer. The loaned sample is returned via this sample.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

See also

FooDataWriter_discard_loan (p. 510)

5.17.4.22 FooDataWriter_discard_loan()

```
DDS_ReturnCode_t FooDataWriter_discard_loan (
    FooDataWriter * self,
    Foo * sample )
```

Returns a loaned sample back to the DataWriter.

This operation is supported while using **Zero Copy transfer** (p. 210) over shared memory" or the **FlatData language binding** (p. 209).

A loaned sample that hasn't been written can be returned to the DataWriter with this operation.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>sample</i>	<< <i>in</i> >> (p. 1013) loaned sample to be discarded.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

FooDataWriter_get_loan (p. 508)

5.17.4.23 DDS_OfferedDeadlineMissedStatus_initialize()

```
DDS_ReturnCode_t DDS_OfferedDeadlineMissedStatus_initialize (
    struct DDS_OfferedDeadlineMissedStatus * self )
```

Initializer for new status instances.

New **DDS_OfferedDeadlineMissedStatus** (p. 1861) instance on heap should be initialized with this function before they are passed to any function. This step ensures that those contained fields that use dynamic memory are properly initialized. This function does not allocate memory.

DDS_OfferedDeadlineMissedStatus_finalize (p. 512) should be called to free the contained fields that use dynamic memory:

```
DDS_OfferedDeadlineMissedStatus *myStatus = malloc(sizeof(struct DDS_OfferedDeadlineMissedStatus));

DDS_OfferedDeadlineMissedStatus_initialize(myStatus);

DDS_DataWriter_get_offered_deadline_missed_status(myDataWriter, myStatus);

DDS_OfferedDeadlineMissedStatus_finalize(myStatus);

free(myStatus);
```

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataWriter_get_offered_deadline_missed_status (p. 546)

DDS_OfferedDeadlineMissedStatus_finalize (p. 512)

5.17.4.24 DDS_OfferedDeadlineMissedStatus_copy()

```
DDS_ReturnCode_t DDS_OfferedDeadlineMissedStatus_copy (
    struct DDS_OfferedDeadlineMissedStatus * self,
    const struct DDS_OfferedDeadlineMissedStatus * source )
```

Copy the contents of the given status into this status.

Status instances can use dynamic memory because of the sequences contained in some status. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). Status to be copied from.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_OfferedDeadlineMissedStatus_INITIALIZER (p. 479)

DDS_OfferedDeadlineMissedStatus_initialize (p. 510)

DDS_OfferedDeadlineMissedStatus_finalize (p. 512)

5.17.4.25 DDS_OfferedDeadlineMissedStatus_finalize()

```
DDS_ReturnCode_t DDS_OfferedDeadlineMissedStatus_finalize (
    struct DDS_OfferedDeadlineMissedStatus * self )
```

Free any dynamic memory allocated by status instances.

some status may use dynamic memory (regardless of whether the status itself is in dynamic memory). This function frees that memory but otherwise leaves this status unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

Note that if this status instance is stored in heap memory, calling this function will *not* call free() on it; the user is responsible for explicitly freeing any heap-based status instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_OfferedDeadlineMissedStatus_INITIALIZER (p. 479)

DDS_OfferedDeadlineMissedStatus_initialize (p. 510)

5.17.4.26 DDS_OfferedDeadlineMissedStatus_equals()

```
DDS_Boolean DDS_OfferedDeadlineMissedStatus_equals (
    const struct DDS_OfferedDeadlineMissedStatus * left,
    const struct DDS_OfferedDeadlineMissedStatus * right )
```

Compares two **DDS_OfferedDeadlineMissedStatus** (p. 1861) for equality.

Parameters

<i>left</i>	<< <i>in</i> >> (p. 1013) This OfferedDeadlineMissedStatus. Can be NULL.
<i>right</i>	<< <i>in</i> >> (p. 1013) The other OfferedDeadlineMissedStatus to be compared with this status instance. Can be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two OfferedDeadlineMissedStatus have equal values, or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.17.4.27 DDS_LivelinessLostStatus_initialize()

```
DDS_ReturnCode_t DDS_LivelinessLostStatus_initialize (
    struct DDS_LivelinessLostStatus * self )
```

Initializer for new status instances.

New **DDS_LivelinessLostStatus** (p. 1822) instance on heap should be initialized with this function before they are passed to any function. This step ensures that those contained fields that use dynamic memory are properly initialized. This function does not allocate memory.

DDS_LivelinessLostStatus_finalize (p. 514) should be called to free the contained fields that use dynamic memory:

```
DDS_LivelinessLostStatus *myStatus = malloc(sizeof(struct DDS_LivelinessLostStatus));
```

```
DDS_LivelinessLostStatus_initialize(myStatus);
```

```
DDS_DataWriter_get_liveliness_lost_status(myDataWriter, myStatus);
```

```
DDS_LivelinessLostStatus_finalize(myStatus);
```

```
free(myStatus);
```

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataWriter_get_liveliness_lost_status (p. 545)

DDS_LivelinessLostStatus_finalize (p. 514)

5.17.4.28 DDS_LivelinessLostStatus_copy()

```
DDS_ReturnCode_t DDS_LivelinessLostStatus_copy (
    struct DDS_LivelinessLostStatus * self,
    const struct DDS_LivelinessLostStatus * source )
```

Copy the contents of the given status into this status.

Status instances can use dynamic memory because of the sequences contained in some status. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). Status to be copied from.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_LivelinessLostStatus_INITIALIZER (p. 480)

DDS_LivelinessLostStatus_initialize (p. 513)

DDS_LivelinessLostStatus_finalize (p. 514)

5.17.4.29 DDS_LivelinessLostStatus_finalize()

```
DDS_ReturnCode_t DDS_LivelinessLostStatus_finalize (
    struct DDS_LivelinessLostStatus * self )
```

Free any dynamic memory allocated by status instances.

some status may use dynamic memory (regardless of whether the status itself is in dynamic memory). This function frees that memory but otherwise leaves this status unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

Note that if this status instance is stored in heap memory, calling this function will *not* call free() on it; the user is responsible for explicitly freeing any heap-based status instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_LivelinessLostStatus_INITIALIZER (p. 480)

DDS_LivelinessLostStatus_initialize (p. 513)

5.17.4.30 DDS_LivelinessLostStatus_equals()

```
DDS_Boolean DDS_LivelinessLostStatus_equals (
    const struct DDS_LivelinessLostStatus * left,
    const struct DDS_LivelinessLostStatus * right )
```

Compares two **DDS_LivelinessLostStatus** (p. 1822) for equality.

Parameters

<i>left</i>	<< <i>in</i> >> (p. 1013) This LivelinessLostStatus. Can be NULL.
<i>right</i>	<< <i>in</i> >> (p. 1013) The other LivelinessLostStatus to be compared with this status instance. Can be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two LivelinessLostStatus have equal values, or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.17.4.31 DDS_OfferedIncompatibleQosStatus_initialize()

```
DDS_ReturnCode_t DDS_OfferedIncompatibleQosStatus_initialize (
    struct DDS_OfferedIncompatibleQosStatus * self )
```

Initializer for new status instances.

New **DDS_OfferedIncompatibleQosStatus** (p. 1862) instance on heap should be initialized with this function before they are passed to any function. This step ensures that those contained fields that use dynamic memory are properly initialized. This function does not allocate memory.

DDS_OfferedIncompatibleQosStatus_finalize (p. 516) should be called to free the contained fields that use dynamic memory:

```
DDS_OfferedIncompatibleQosStatus *myStatus = malloc(sizeof(struct DDS_OfferedIncompatibleQosStatus));

DDS_OfferedIncompatibleQosStatus_initialize(myStatus);

DDS_DataWriter_get_offered_incompatible_qos_status(myDataWriter, myStatus);

DDS_OfferedIncompatibleQosStatus_finalize(myStatus);

free(myStatus);
```

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataWriter_get_offered_incompatible_qos_status (p. 546)

DDS_OfferedIncompatibleQosStatus_finalize (p. 516)

5.17.4.32 DDS_OfferedIncompatibleQosStatus_copy()

```
DDS_ReturnCode_t DDS_OfferedIncompatibleQosStatus_copy (
    struct DDS_OfferedIncompatibleQosStatus * self,
    const struct DDS_OfferedIncompatibleQosStatus * source )
```

Copy the contents of the given status into this status.

Status instances can use dynamic memory because of the sequences contained in some status. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). Status to be copied from.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_OfferedIncompatibleQosStatus_INITIALIZER (p. 480)

DDS_OfferedIncompatibleQosStatus_initialize (p. 515)

DDS_OfferedIncompatibleQosStatus_finalize (p. 516)

5.17.4.33 DDS_OfferedIncompatibleQosStatus_finalize()

```
DDS_ReturnCode_t DDS_OfferedIncompatibleQosStatus_finalize (
    struct DDS_OfferedIncompatibleQosStatus * self )
```

Free any dynamic memory allocated by status instances.

some status may use dynamic memory (regardless of whether the status itself is in dynamic memory). This function frees that memory but otherwise leaves this status unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

Note that if this status instance is stored in heap memory, calling this function will *not* call free() on it; the user is responsible for explicitly freeing any heap-based status instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_OfferedIncompatibleQosStatus_INITIALIZER (p. 480)

DDS_OfferedIncompatibleQosStatus_initialize (p. 515)

5.17.4.34 DDS_OfferedIncompatibleQosStatus_equals()

```
DDS_Boolean DDS_OfferedIncompatibleQosStatus_equals (
    const struct DDS_OfferedIncompatibleQosStatus * left,
    const struct DDS_OfferedIncompatibleQosStatus * right )
```

Compares two **DDS_OfferedIncompatibleQosStatus** (p. 1862) for equality.

Parameters

<i>left</i>	<< <i>in</i> >> (p. 1013) This OfferedIncompatibleQosStatus. Can be NULL.
<i>right</i>	<< <i>in</i> >> (p. 1013) The other OfferedIncompatibleQosStatus to be compared with this status instance. Can be NULL.

Returns

DDS_BOOLEAN_TRUE (p.1201) if the two OfferedIncompatibleQosStatus have equal values, or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.17.4.35 DDS_PublicationMatchedStatus_initialize()

```
DDS_ReturnCode_t DDS_PublicationMatchedStatus_initialize (
    struct DDS_PublicationMatchedStatus * self )
```

Initializer for new status instances.

New **DDS_PublicationMatchedStatus** (p. 1914) instance on heap should be initialized with this function before they are passed to any function. This step ensures that those contained fields that use dynamic memory are properly initialized. This function does not allocate memory.

DDS_PublicationMatchedStatus_finalize (p. 519) should be called to free the contained fields that use dynamic memory:

```
DDS_PublicationMatchedStatus *myStatus = malloc(sizeof(struct DDS_PublicationMatchedStatus));
DDS_PublicationMatchedStatus_initialize(myStatus);
DDS_DataWriter_get_publication_matched_status(myDataWriter, myStatus);
DDS_PublicationMatchedStatus_finalize(myStatus);
free(myStatus);
```

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataWriter_get_publication_matched_status (p. 546)

DDS_PublicationMatchedStatus_finalize (p. 519)

5.17.4.36 DDS_PublicationMatchedStatus_copy()

```
DDS_ReturnCode_t DDS_PublicationMatchedStatus_copy (
    struct DDS_PublicationMatchedStatus * self,
    const struct DDS_PublicationMatchedStatus * source )
```

Copy the contents of the given status into this status.

Status instances can use dynamic memory because of the sequences contained in some status. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). Status to be copied from.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_PublicationMatchedStatus_INITIALIZER (p. 481)

DDS_PublicationMatchedStatus_initialize (p. 518)

DDS_PublicationMatchedStatus_finalize (p. 519)

5.17.4.37 DDS_PublicationMatchedStatus_finalize()

```
DDS_ReturnCode_t DDS_PublicationMatchedStatus_finalize (
    struct DDS_PublicationMatchedStatus * self )
```

Free any dynamic memory allocated by status instances.

some status may use dynamic memory (regardless of whether the status itself is in dynamic memory). This function frees that memory but otherwise leaves this status unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

Note that if this status instance is stored in heap memory, calling this function will *not* call free() on it; the user is responsible for explicitly freeing any heap-based status instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_PublicationMatchedStatus_INITIALIZER (p. 481)

DDS_PublicationMatchedStatus_initialize (p. 518)

5.17.4.38 DDS_PublicationMatchedStatus_equals()

```
DDS_Boolean DDS_PublicationMatchedStatus_equals (
    const struct DDS_PublicationMatchedStatus * left,
    const struct DDS_PublicationMatchedStatus * right )
```

Compares two **DDS_PublicationMatchedStatus** (p. 1914) for equality.

Parameters

<i>left</i>	<< <i>in</i> >> (p. 1013) This PublicationMatchedStatus. Can be NULL.
<i>right</i>	<< <i>in</i> >> (p. 1013) The other PublicationMatchedStatus to be compared with this status instance. Can be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two PublicationMatchedStatus have equal values, or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.17.4.39 DDS_ServiceRequestAcceptedStatus_initialize()

```
DDS_ReturnCode_t DDS_ServiceRequestAcceptedStatus_initialize (
    struct DDS_ServiceRequestAcceptedStatus * self )
```

Initializer for new status instances.

New **DDS_ServiceRequestAcceptedStatus** (p. 2002) instance on heap should be initialized with this function before they are passed to any function. This step ensures that those contained fields that use dynamic memory are properly initialized. This function does not allocate memory.

DDS_ServiceRequestAcceptedStatus_finalize (p. 521) should be called to free the contained fields that use dynamic memory:

```
DDS_ServiceRequestAcceptedStatus *myStatus = malloc(sizeof(struct DDS_ServiceRequestAcceptedStatus));
DDS_ServiceRequestAcceptedStatus_initialize(myStatus);
DDS_DataWriter_get_service_request_accepted_status(myDataWriter, myStatus);
DDS_ServiceRequestAcceptedStatus_finalize(myStatus);
free(myStatus);
```

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataWriter_get_service_request_accepted_status (p. 550)

DDS_ServiceRequestAcceptedStatus_finalize (p. 521)

5.17.4.40 DDS_ServiceRequestAcceptedStatus_copy()

```
DDS_ReturnCode_t DDS_ServiceRequestAcceptedStatus_copy (
    struct DDS_ServiceRequestAcceptedStatus * self,
    const struct DDS_ServiceRequestAcceptedStatus * source )
```

Copy the contents of the given status into this status.

Status instances can use dynamic memory because of the sequences contained in some status. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). Status to be copied from.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_ServiceRequestAcceptedStatus_INITIALIZER (p. 481)

DDS_ServiceRequestAcceptedStatus_initialize (p. 520)

DDS_ServiceRequestAcceptedStatus_finalize (p. 521)

5.17.4.41 DDS_ServiceRequestAcceptedStatus_finalize()

```
DDS_ReturnCode_t DDS_ServiceRequestAcceptedStatus_finalize (
    struct DDS_ServiceRequestAcceptedStatus * self )
```

Free any dynamic memory allocated by status instances.

some status may use dynamic memory (regardless of whether the status itself is in dynamic memory). This function frees that memory but otherwise leaves this status unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

Note that if this status instance is stored in heap memory, calling this function will *not* call free() on it; the user is responsible for explicitly freeing any heap-based status instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_ServiceRequestAcceptedStatus_INITIALIZER (p. 481)

DDS_ServiceRequestAcceptedStatus_initialize (p. 520)

5.17.4.42 DDS_ServiceRequestAcceptedStatus_equals()

```
DDS_Boolean DDS_ServiceRequestAcceptedStatus_equals (
    const struct DDS_ServiceRequestAcceptedStatus * left,
    const struct DDS_ServiceRequestAcceptedStatus * right )
```

Compares two **DDS_ServiceRequestAcceptedStatus** (p. 2002) for equality.

Parameters

<i>left</i>	<< <i>in</i> >> (p. 1013) This ServiceRequestAcceptedStatus. Can be NULL.
<i>right</i>	<< <i>in</i> >> (p. 1013) The other ServiceRequestAcceptedStatus to be compared with this status instance. Can be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two ServiceRequestAcceptedStatus have equal values, or **DDS_↵
BOOLEAN_FALSE** (p. 1201) otherwise.

5.17.4.43 DDS_ReliableWriterCacheEventCount_equals()

```
DDS_Boolean DDS_ReliableWriterCacheEventCount_equals (
    const struct DDS_ReliableWriterCacheEventCount * left,
    const struct DDS_ReliableWriterCacheEventCount * right )
```

Compares two **DDS_ReliableWriterCacheEventCount** (p. 1945) for equality.

Parameters

<i>left</i>	<< <i>in</i> >> (p. 1013) This ReliableWriterCacheEventCount. Can be NULL.
<i>right</i>	<< <i>in</i> >> (p. 1013) The other ReliableWriterCacheEventCount to be compared with this status instance. Can be NULL.

Returns

DDS_BOOLEAN_TRUE (p.1201) if the two ReliableWriterCacheEventCount have equal values, or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.17.4.44 DDS_ReliableWriterCacheChangedStatus_initialize()

```
DDS_ReturnCode_t DDS_ReliableWriterCacheChangedStatus_initialize (
    struct DDS_ReliableWriterCacheChangedStatus * self )
```

Initializer for new status instances.

New **DDS_ReliableWriterCacheChangedStatus** (p.1942) instance on heap should be initialized with this function before they are passed to any function. This step ensures that those contained fields that use dynamic memory are properly initialized. This function does not allocate memory.

DDS_ReliableWriterCacheChangedStatus_finalize (p. 524) should be called to free the contained fields that use dynamic memory:

```
DDS_ReliableWriterCacheChangedStatus *myStatus = malloc(sizeof(struct DDS_ReliableWriterCacheChangedStatus));

DDS_ReliableWriterCacheChangedStatus_initialize(myStatus);

DDS_DataWriter_get_reliable_writer_cache_changed_status(myDataWriter, myStatus);

DDS_ReliableWriterCacheChangedStatus_finalize(myStatus);

free(myStatus);
```

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataWriter_get_reliable_writer_cache_changed_status (p.547)

DDS_ReliableWriterCacheChangedStatus_finalize (p. 524)

5.17.4.45 DDS_ReliableWriterCacheChangedStatus_copy()

```
DDS_ReturnCode_t DDS_ReliableWriterCacheChangedStatus_copy (
    struct DDS_ReliableWriterCacheChangedStatus * self,
    const struct DDS_ReliableWriterCacheChangedStatus * source )
```

Copy the contents of the given status into this status.

Status instances can use dynamic memory because of the sequences contained in some status. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). Status to be copied from.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_ReliableWriterCacheChangedStatus_INITIALIZER (p. 482)

DDS_ReliableWriterCacheChangedStatus_initialize (p. 523)

DDS_ReliableWriterCacheChangedStatus_finalize (p. 524)

5.17.4.46 DDS_ReliableWriterCacheChangedStatus_finalize()

```
DDS_ReturnCode_t DDS_ReliableWriterCacheChangedStatus_finalize (
    struct DDS_ReliableWriterCacheChangedStatus * self )
```

Free any dynamic memory allocated by status instances.

some status may use dynamic memory (regardless of whether the status itself is in dynamic memory). This function frees that memory but otherwise leaves this status unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

Note that if this status instance is stored in heap memory, calling this function will *not* call free() on it; the user is responsible for explicitly freeing any heap-based status instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_ReliableWriterCacheChangedStatus_INITIALIZER (p. 482)

DDS_ReliableWriterCacheChangedStatus_initialize (p. 523)

5.17.4.47 DDS_ReliableWriterCacheChangedStatus_equals()

```
DDS_Boolean DDS_ReliableWriterCacheChangedStatus_equals (
    const struct DDS_ReliableWriterCacheChangedStatus * left,
    const struct DDS_ReliableWriterCacheChangedStatus * right )
```

Compares two **DDS_ReliableWriterCacheChangedStatus** (p. 1942) for equality.

Parameters

<i>left</i>	<< <i>in</i> >> (p. 1013) This ReliableWriterCacheChangedStatus. Can be NULL.
<i>right</i>	<< <i>in</i> >> (p. 1013) The other ReliableWriterCacheChangedStatus to be compared with this status instance. Can be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two ReliableWriterCacheChangedStatus have equal values, or **DDS_↵
BOOLEAN_FALSE** (p. 1201) otherwise.

5.17.4.48 DDS_ReliableReaderActivityChangedStatus_initialize()

```
DDS_ReturnCode_t DDS_ReliableReaderActivityChangedStatus_initialize (
    struct DDS_ReliableReaderActivityChangedStatus * self )
```

Initializer for new status instances.

New **DDS_ReliableReaderActivityChangedStatus** (p. 1940) instance on heap should be initialized with this function before they are passed to any function. This step ensures that those contained fields that use dynamic memory are properly initialized. This function does not allocate memory.

DDS_ReliableReaderActivityChangedStatus_finalize (p. 526) should be called to free the contained fields that use dynamic memory:

```
DDS_ReliableReaderActivityChangedStatus *myStatus = malloc(sizeof(struct
    DDS_ReliableReaderActivityChangedStatus));

DDS_ReliableReaderActivityChangedStatus_initialize(myStatus);

DDS_DataWriter_get_reliable_reader_activity_changed_status(myDataWriter, myStatus);

DDS_ReliableReaderActivityChangedStatus_finalize(myStatus);

free(myStatus);
```

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataWriter_get_reliable_reader_activity_changed_status (p. 547)

DDS_ReliableReaderActivityChangedStatus_finalize (p. 526)

5.17.4.49 DDS_ReliableReaderActivityChangedStatus_copy()

```
DDS_ReturnCode_t DDS_ReliableReaderActivityChangedStatus_copy (
    struct DDS_ReliableReaderActivityChangedStatus * self,
    const struct DDS_ReliableReaderActivityChangedStatus * source )
```

Copy the contents of the given status into this status.

Status instances can use dynamic memory because of the sequences contained in some status. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). Status to be copied from.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_ReliableReaderActivityChangedStatus_INITIALIZER (p. 482)

DDS_ReliableReaderActivityChangedStatus_initialize (p. 525)

DDS_ReliableReaderActivityChangedStatus_finalize (p. 526)

5.17.4.50 DDS_ReliableReaderActivityChangedStatus_finalize()

```
DDS_ReturnCode_t DDS_ReliableReaderActivityChangedStatus_finalize (
    struct DDS_ReliableReaderActivityChangedStatus * self )
```

Free any dynamic memory allocated by status instances.

some status may use dynamic memory (regardless of whether the status itself is in dynamic memory). This function frees that memory but otherwise leaves this status unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

Note that if this status instance is stored in heap memory, calling this function will *not* call `free()` on it; the user is responsible for explicitly freeing any heap-based status instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_ReliableReaderActivityChangedStatus_INITIALIZER (p. 482)

DDS_ReliableReaderActivityChangedStatus_initialize (p. 525)

5.17.4.51 DDS_ReliableReaderActivityChangedStatus_equals()

```
DDS_Boolean DDS_ReliableReaderActivityChangedStatus_equals (
    const struct DDS_ReliableReaderActivityChangedStatus * left,
    const struct DDS_ReliableReaderActivityChangedStatus * right )
```

Compares two **DDS_ReliableReaderActivityChangedStatus** (p. 1940) for equality.

Parameters

<i>left</i>	<< <i>in</i> >> (p. 1013) This ReliableReaderActivityChangedStatus. Can be NULL.
<i>right</i>	<< <i>in</i> >> (p. 1013) The other ReliableReaderActivityChangedStatus to be compared with this status instance. Can be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two ReliableReaderActivityChangedStatus have equal values, or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.17.4.52 DDS_DataWriterCacheStatus_initialize()

```
DDS_ReturnCode_t DDS_DataWriterCacheStatus_initialize (
    struct DDS_DataWriterCacheStatus * self )
```

Initializer for new status instances.

New **DDS_DataWriterCacheStatus** (p. 1641) instance on heap should be initialized with this function before they are passed to any function. This step ensures that those contained fields that use dynamic memory are properly initialized. This function does not allocate memory.

DDS_DataWriterCacheStatus_finalize (p.529) should be called to free the contained fields that use dynamic memory:

```
DDS_DataWriterCacheStatus *myStatus = malloc(sizeof(struct DDS_DataWriterCacheStatus));

DDS_DataWriterCacheStatus_initialize(myStatus);

DDS_DataWriter_get_datawriter_cache_status(myDataWriter, myStatus);

DDS_DataWriterCacheStatus_finalize(myStatus);

free(myStatus);
```

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataWriter_get_datawriter_cache_status (p. 548)

DDS_DataWriterCacheStatus_finalize (p. 529)

5.17.4.53 DDS_DataWriterCacheStatus_copy()

```
DDS_ReturnCode_t DDS_DataWriterCacheStatus_copy (
    struct DDS_DataWriterCacheStatus * self,
    const struct DDS_DataWriterCacheStatus * source )
```

Copy the contents of the given status into this status.

Status instances can use dynamic memory because of the sequences contained in some status. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). Status to be copied from.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataWriterCacheStatus_INITIALIZER (p. 483)

DDS_DataWriterCacheStatus_initialize (p. 528)

DDS_DataWriterCacheStatus_finalize (p. 529)

5.17.4.54 DDS_DataWriterCacheStatus_finalize()

```
DDS_ReturnCode_t DDS_DataWriterCacheStatus_finalize (
    struct DDS_DataWriterCacheStatus * self )
```

Free any dynamic memory allocated by status instances.

some status may use dynamic memory (regardless of whether the status itself is in dynamic memory). This function frees that memory but otherwise leaves this status unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

Note that if this status instance is stored in heap memory, calling this function will *not* call free() on it; the user is responsible for explicitly freeing any heap-based status instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataWriterCacheStatus_INITIALIZER (p. 483)

DDS_DataWriterCacheStatus_initialize (p. 528)

5.17.4.55 DDS_DataWriterCacheStatus_equals()

```
DDS_Boolean DDS_DataWriterCacheStatus_equals (
    const struct DDS_DataWriterCacheStatus * left,
    const struct DDS_DataWriterCacheStatus * right )
```

Compares two **DDS_DataWriterCacheStatus** (p. 1641) for equality.

Parameters

<i>left</i>	<< <i>in</i> >> (p. 1013) This DataWriterCacheStatus. Can be NULL.
<i>right</i>	<< <i>in</i> >> (p. 1013) The other DataWriterCacheStatus to be compared with this status instance. Can be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two DataWriterCacheStatus have equal values, or **DDS_BOOLEAN_↵**
FALSE (p. 1201) otherwise.

5.17.4.56 DDS_DataWriterProtocolStatus_initialize()

```
DDS_ReturnCode_t DDS_DataWriterProtocolStatus_initialize (
    struct DDS_DataWriterProtocolStatus * self )
```

Initializer for new status instances.

New **DDS_DataWriterProtocolStatus** (p. 1654) instance on heap should be initialized with this function before they are passed to any function. This step ensures that those contained fields that use dynamic memory are properly initialized. This function does not allocate memory.

DDS_DataWriterProtocolStatus_finalize (p. 532) should be called to free the contained fields that use dynamic memory:

```
DDS_DataWriterProtocolStatus *myStatus = malloc(sizeof(struct DDS_DataWriterProtocolStatus));
DDS_DataWriterProtocolStatus_initialize(myStatus);
DDS_DataWriter_get_datawriter_protocol_status(myDataWriter, myStatus);
DDS_DataWriterProtocolStatus_finalize(myStatus);
free(myStatus);
```

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataWriter_get_datawriter_protocol_status (p. 548)

DDS_DataWriterProtocolStatus_finalize (p. 532)

5.17.4.57 DDS_DataWriterProtocolStatus_copy()

```
DDS_ReturnCode_t DDS_DataWriterProtocolStatus_copy (
    struct DDS_DataWriterProtocolStatus * self,
    const struct DDS_DataWriterProtocolStatus * source )
```

Copy the contents of the given status into this status.

Status instances can use dynamic memory because of the sequences contained in some status. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). Status to be copied from.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataWriterProtocolStatus_INITIALIZER (p. 483)

DDS_DataWriterProtocolStatus_initialize (p. 531)

DDS_DataWriterProtocolStatus_finalize (p. 532)

5.17.4.58 DDS_DataWriterProtocolStatus_finalize()

```
DDS_ReturnCode_t DDS_DataWriterProtocolStatus_finalize (
    struct DDS_DataWriterProtocolStatus * self )
```

Free any dynamic memory allocated by status instances.

some status may use dynamic memory (regardless of whether the status itself is in dynamic memory). This function frees that memory but otherwise leaves this status unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

Note that if this status instance is stored in heap memory, calling this function will *not* call free() on it; the user is responsible for explicitly freeing any heap-based status instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataWriterProtocolStatus_INITIALIZER (p. 483)

DDS_DataWriterProtocolStatus_initialize (p. 531)

5.17.4.59 DDS_DataWriterProtocolStatus_equals()

```
DDS_Boolean DDS_DataWriterProtocolStatus_equals (
    const struct DDS_DataWriterProtocolStatus * left,
    const struct DDS_DataWriterProtocolStatus * right )
```

Compares two **DDS_DataWriterProtocolStatus** (p. 1654) for equality.

Parameters

<i>left</i>	<< <i>in</i> >> (p. 1013) This DataWriterProtocolStatus. Can be NULL.
<i>right</i>	<< <i>in</i> >> (p. 1013) The other DataWriterProtocolStatus to be compared with this status instance. Can be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two DataWriterProtocolStatus have equal values, or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.17.4.60 DDS_DataWriterQos_equals()

```
DDS_Boolean DDS_DataWriterQos_equals (
    const struct DDS_DataWriterQos * self,
    const struct DDS_DataWriterQos * other )
```

Compares two **DDS_DataWriterQos** (p. 1666) for equality.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) This DataWriterQos.
<i>other</i>	<< <i>in</i> >> (p. 1013) The other DataWriterQos to be compared with this DataWriterQos.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two Qos objects are equal or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.17.4.61 DDS_DataWriterQos_print()

```
DDS_ReturnCode_t DDS_DataWriterQos_print (
    const struct DDS_DataWriterQos * self )
```

Prints this **DDS_DataWriterQos** (p. 1666) to stdout.

Only the differences between this **DDS_DataWriterQos** (p. 1666) and the documented default are printed. If you wish to print everything regardless, see **DDS_DataWriterQos_to_string_w_params** (p. 534). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1926).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_RETCODE_OK (p. 1222) if no error was encountered.

5.17.4.62 DDS_DataWriterQos_to_string()

```
DDS_ReturnCode_t DDS_DataWriterQos_to_string (
    const struct DDS_DataWriterQos * self,
    char * string,
    DDS_UnsignedLong * string_size )
```

Obtains a string representation of this **DDS_DataWriterQos** (p. 1666).

Only the differences between this **DDS_DataWriterQos** (p. 1666) and the documented default are printed to the string. If you wish to print everything regardless, see **DDS_DataWriterQos_to_string_w_params** (p. 534). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1926).

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. 1666) is written to the buffer.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>string</i>	<< <i>out</i> >> (p. 1013) The char buffer that will be used to store the string representation of this DDS_DataWriterQos (p. 1666). If NULL, this function will return the required length of this buffer through the <i>string_size</i> parameter.
<i>string_size</i>	<< <i>inout</i> >> (p. 1013) 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. 1222) if no error was encountered.

See also

DDS_DataWriterQos_to_string_w_params (p. 534)

5.17.4.63 DDS_DataWriterQos_to_string_w_params()

```
DDS_ReturnCode_t DDS_DataWriterQos_to_string_w_params (
    const struct DDS_DataWriterQos * self,
    char * string,
    DDS_UnsignedLong * string_size,
    const struct DDS_DataWriterQos * base,
    const struct DDS_QosPrintFormat * format )
```

Obtains a string representation of this **DDS_DataWriterQos** (p. 1666).

Only the differences between this **DDS_DataWriterQos** (p. 1666) and the **DDS_DataWriterQos** (p. 1666) supplied as the base are printed to the string.

It is possible to supply the sentinel value **DDS_DATAWRITER_QOS_PRINT_ALL** (p. 469) as the base to print everything within the QoS.

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. 1666) is written to the buffer.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>string</i>	<< <i>out</i> >> (p. 1013) The char buffer that will be used to store the string representation of this DDS_DataWriterQos (p. 1666). If NULL, this function will return the required length of this buffer through the string_size parameter.
<i>string_size</i>	<< <i>inout</i> >> (p. 1013) 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.
<i>base</i>	<< <i>in</i> >> (p. 1013) The DDS_DataWriterQos (p. 1666) to be used as the base QoS profile. Only the differences with respect to this base profile will be included in the output string.
<i>format</i>	<< <i>in</i> >> (p. 1013) The DDS_QosPrintFormat (p. 1926) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 1222) if no error was encountered.

5.17.4.64 DDS_DataWriterQos_initialize()

```
DDS_ReturnCode_t DDS_DataWriterQos_initialize (
    struct DDS_DataWriterQos * self )
```

Initializer for new QoS instances.

New **DDS_DataWriterQos** (p. 1666) instances on heap should be initialized with this function before they are passed to any function. This step ensures that those contained QoS policies that use dynamic memory are properly initialized. This function does not allocate memory.

Calling this function is not a substitute for calling **DDS_DataWriter_get_qos** (p. 552) or **DDS_Publisher_get_default_datawriter_qos** (p. 449); one of those should be called subsequently to setting the QoS of any new or existing entity. **DDS_DataWriterQos_finalize** (p. 536) should be called to free the contained QoS policies that use dynamic memory:

```
DDS_DataWriterQos *myQos = malloc(sizeof(struct DDS_DataWriterQos));

DDS_DataWriterQos_initialize(myQos);

DDS_Publisher_get_default_datawriter_qos(myFactory, myQos);

DDS_DataWriter_set_qos(myDataWriter, myQos);

DDS_DataWriterQos_finalize(myQos);

free(myQos);
```

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_Publisher_get_default_datawriter_qos (p. 449)

DDS_DataWriterQos_finalize (p. 536)

5.17.4.65 DDS_DataWriterQos_finalize()

```
DDS_ReturnCode_t DDS_DataWriterQos_finalize (
    struct DDS_DataWriterQos * self )
```

Free any dynamic memory allocated by the policies in this **DDS_DataWriterQos** (p. 1666).

Some QoS policies may use dynamic memory (regardless of whether the QoS itself is in dynamic memory). This function frees that memory but otherwise leaves this QoS unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

This function does not leave this object in an invalid state. It is permissible to clear a QoS and then subsequently allocate new dynamic memory in one or more of its QoS policies.

Note that if this QoS instance is stored in heap memory, calling this function will *not* call `free()` on it; the user is responsible for explicitly freeing any heap-based QoS instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataWriterQos_INITIALIZER (p. 484)

DDS_DataWriterQos_initialize (p. 535)

5.17.4.66 DDS_DataWriterQos_copy()

```
DDS_ReturnCode_t DDS_DataWriterQos_copy (
    struct DDS_DataWriterQos * self,
    const struct DDS_DataWriterQos * source )
```

Copy the contents of the given QoS into this QoS.

DDS_DataWriterQos (p. 1666) instances can use dynamic memory because of the sequences contained in some QoS policies. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013) The DDS_DataWriterQos (p. 1666) to copy from.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataWriterQos_INITIALIZER (p. 484)

DDS_DataWriterQos_initialize (p. 535)

DDS_DataWriterQos_finalize (p. 536)

5.17.4.67 DDS_DataWriter_as_entity()

```
DDS_Entity * DDS_DataWriter_as_entity (
    DDS_DataWriter * dataWriter )
```

Access a **DDS_DataWriter** (p. 485)'s supertype instance.

Parameters

<i>dataWriter</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

The **DDS_Entity** (p. 1360) that is supertype instance of the datawriter.

5.17.4.68 **DDS_DataWriter_assert_liveliness()**

```
DDS_ReturnCode_t DDS_DataWriter_assert_liveliness (
    DDS_DataWriter * self )
```

This operation manually asserts the liveliness of this **DDS_DataWriter** (p. 485).

This is used in combination with the **LIVELINESS** (p. 1296) policy to indicate to RTI Connex that the **DDS_DataWriter** (p. 485) remains active.

You only need to use this operation if the **LIVELINESS** (p. 1296) setting is either **DDS_MANUAL_BY_PARTICIPANT** ↔ **LIVELINESS_QOS** (p. 1297) or **DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS** (p. 1297). Otherwise, it has no effect.

Note: writing data via the **FooDataWriter_write** (p. 496) or **FooDataWriter_write_w_timestamp** (p. 500) operation asserts liveliness on the **DDS_DataWriter** (p. 485) itself, and its **DDS_DomainParticipant** (p. 77). Consequently the use of `assert_liveliness()` is only needed if the application is not writing data regularly.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_NOT_ENABLED** (p. 1222)

See also

DDS_LivelinessQosPolicy (p. 1823)

5.17.4.69 **DDS_DataWriter_get_matched_subscription_locators()**

```
DDS_ReturnCode_t DDS_DataWriter_get_matched_subscription_locators (
    DDS_DataWriter * self,
    struct DDS_LocatorSeq * locators )
```

<<*extension*>> (p. 1012) Retrieve the list of locators for subscriptions currently "associated" with this **DDS_DataWriter** (p. 485).

The locators returned in the `locators` list are the ones that are used by the DDS implementation to communicate with the corresponding matched **DDS_DataReader** (p. 615) entities.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>locators</i>	<< <i>inout</i> >> (p. 1013). 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 function will fail with **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222). Cannot be NULL. .

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222) if the sequence is too small and the system can not resize it, or **DDS_RETCODE_NOT_ENABLED** (p. 1222)

5.17.4.70 DDS_DataWriter_get_matched_subscriptions()

```
DDS_ReturnCode_t DDS_DataWriter_get_matched_subscriptions (
    DDS_DataWriter * self,
    struct DDS_InstanceHandleSeq * subscription_handles )
```

Retrieve the list of subscriptions currently "associated" with this **DDS_DataWriter** (p. 485).

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 **DDS_Topic** (p. 175).
- The subscription has compatible QoS.
- If the applications are using partitions, the subscription shares a common partition with this publication.
- The **DDS_DomainParticipant** (p. 77) has not indicated that the subscription's **DDS_DomainParticipant** (p. 77) should be "ignored" by means of the **DDS_DomainParticipant_ignore_publication** (p. 132) API.
- If the publication is using the **DDS_MultiChannelQosPolicy** (p. 1856) and the subscription is using a **DDS_ContentFilteredTopic** (p. 177), 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 **DDS_DataReader** (p. 615) entities. These handles match the ones that appear in the **DDS_SampleInfo::instance_handle** (p. 1990) field of the **DDS_SampleInfo** (p. 1985) when reading the **DDS_SUBSCRIPTION_TOPIC_NAME** (p. 1098) builtin topic.

This API may return the subscription handles of subscriptions that are inactive. **DDS_DataWriter_is_matched_subscription_active** (p. 540) can be used to check this.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>subscription_handles</i>	<< <i>inout</i> >> (p. 1013). 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 function will fail with **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

The maximum number of matches possible is configured with **DDS_DomainParticipantResourceLimitsQosPolicy** (p. 1731). You can use a zero-maximum sequence without ownership to quickly check whether there are any matches without allocating any memory. Cannot be NULL. .

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222) if the sequence is too small and the system cannot resize it, or **DDS_RETCODE_NOT_ENABLED** (p. 1222)

5.17.4.71 DDS_DataWriter_is_matched_subscription_active()

```
DDS_ReturnCode_t DDS_DataWriter_is_matched_subscription_active (
    DDS_DataWriter * self,
    DDS_Boolean * is_active,
    const DDS_InstanceHandle_t * subscription_handle )
```

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 nonprogressing (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. 214) will become invalid and this function will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 1222).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>subscription_handle</i>	<< <i>in</i> >> (p. 1013) The DDS_InstanceHandle_t (p. 214) of the matched subscription. See DDS_DataWriter_get_matched_subscriptions (p. 539) for a description of what is considered a matched subscription.
<i>is_active</i>	<< <i>out</i> >> (p. 1013) Indicates whether or not the matched subscription is active.

Returns

One of the **Standard Return Codes** (p. 1221)

5.17.4.72 DDS_DataWriter_get_matched_subscription_data()

```
DDS_ReturnCode_t DDS_DataWriter_get_matched_subscription_data (
    DDS_DataWriter * self,
    struct DDS_SubscriptionBuiltinTopicData * subscription_data,
    const DDS_InstanceHandle_t * subscription_handle )
```

This operation retrieves the information on a subscription that is currently "associated" with the **DDS_DataWriter** (p. 485).

The `subscription_handle` must correspond to a subscription currently associated with the **DDS_DataWriter** (p. 485). Otherwise, the operation will fail and fail with **DDS_RETCODE_BAD_PARAMETER** (p. 1222). Use **DDS_DataWriter_get_matched_subscriptions** (p. 539) to find the subscriptions that are currently matched with the **DDS_DataWriter** (p. 485).

The above information is also available through **DDS_DataReaderListener::on_data_available()** (p. 1595) (if a reader listener is installed on the **DDS_SubscriptionBuiltinTopicDataDataReader** (p. 1097)).

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 **DDS_DataReaderListener::on_data_available()** (p. 1595) of the **DDS_SubscriptionBuiltinTopicDataDataReader** (p. 1097). 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 **DDS_SubscriptionBuiltinTopicDataDataReader** (p. 1097). 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. 2018). This information is available through **DDS_DataReaderListener::on_data_available()** (p. 1595) (if a reader listener is installed on the **DDS_SubscriptionBuiltinTopicDataDataReader** (p. 1097)).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>subscription_data</i>	<< <i>inout</i> >> (p. 1013). The information to be filled in on the associated subscription. Cannot be NULL.
<i>subscription_handle</i>	<< <i>in</i> >> (p. 1013). Handle to a specific subscription associated with the DDS_DataReader (p. 615). Must correspond to a subscription currently associated with the DDS_DataWriter (p. 485). Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_NOT_ENABLED** (p. 1222), or **DDS_RETCODE_TIMEOUT** (p. 1222)

5.17.4.73 DDS_DataWriter_get_matched_subscription_participant_data()

```
DDS_ReturnCode_t DDS_DataWriter_get_matched_subscription_participant_data (
    DDS_DataWriter * self,
    struct DDS_ParticipantBuiltinTopicData * participant_data,
    const DDS_InstanceHandle_t * subscription_handle )
```

This operation retrieves the information on the discovered **DDS_DomainParticipant** (p. 77) associated with the subscription that is currently matching with the **DDS_DataWriter** (p. 485).

The `subscription_handle` must correspond to a subscription currently associated with the **DDS_DataWriter** (p. 485). Otherwise, the operation will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 1222). The operation may also fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) if the subscription corresponds to the same **DDS_DomainParticipant** (p. 77) that the DataWriter belongs to. Use **DDS_DataWriter_get_matched_subscriptions** (p. 539) to find the subscriptions that are currently matched with the **DDS_DataWriter** (p. 485).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>participant_data</i>	<< <i>inout</i> >> (p. 1013). The information to be filled in on the associated participant Cannot be NULL.
<i>subscription_handle</i>	<< <i>in</i> >> (p. 1013). Handle to a specific subscription associated with the DDS_DataReader (p. 615). Must correspond to a subscription currently associated with the DDS_DataWriter (p. 485).

Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_NOT_ENABLED** (p. 1222)

5.17.4.74 DDS_DataWriter_get_topic()

```
DDS_Topic * DDS_DataWriter_get_topic (
    DDS_DataWriter * self )
```

This operation returns the **DDS_Topic** (p. 175) associated with the **DDS_DataWriter** (p. 485).

This is the same **DDS_Topic** (p. 175) that was used to create the **DDS_DataWriter** (p. 485).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_Topic (p. 175) that was used to create the **DDS_DataWriter** (p. 485).

5.17.4.75 **DDS_DataWriter_get_publisher()**

```
DDS_Publisher * DDS_DataWriter_get_publisher (
    DDS_DataWriter * self )
```

This operation returns the **DDS_Publisher** (p. 443) to which the **DDS_DataWriter** (p. 485) belongs.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_Publisher (p. 443) to which the **DDS_DataWriter** (p. 485) belongs.

5.17.4.76 **DDS_DataWriter_wait_for_acknowledgments()**

```
DDS_ReturnCode_t DDS_DataWriter_wait_for_acknowledgments (
    DDS_DataWriter * self,
    const struct DDS_Duration_t * max_wait )
```

Blocks the calling thread until all data written by reliable **DDS_DataWriter** (p. 485) entity is acknowledged, or until timeout expires.

This operation blocks the calling thread until either all data written by the reliable **DDS_DataWriter** (p. 485) entity is acknowledged by (a) all reliable **DDS_DataReader** (p. 615) 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 **DDS_DataWriter** (p. 485) does not have **DDS_ReliabilityQosPolicy** (p. 1936) kind set to `RELIABLE`, this operation will succeed immediately.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>max_wait</i>	<< <i>in</i> >> (p. 1013) Specifies maximum time to wait for acknowledgements DDS_Duration_t (p. 1767) .

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_NOT_ENABLED** (p. 1222), **DDS_RETCODE_←
_TIMEOUT** (p. 1222)

5.17.4.77 DDS_DataWriter_is_sample_app_acknowledged()

```
DDS_ReturnCode_t DDS_DataWriter_is_sample_app_acknowledged (
    DDS_DataWriter * self,
    DDS_Boolean * is_app_ack,
    const struct DDS_SampleIdentity_t * identity )
```

This function can be used to see if a sample has been application acknowledged.

This function 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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>is_app_ack</i>	<< <i>out</i> >> (p. 1013) This value will be set to DDS_BOOLEAN_TRUE (p. 1201) when the sample has been acknowledged.
<i>identity</i>	<< <i>in</i> >> (p. 1013) Sample identity.

Returns

One of the **Standard Return Codes** (p. 1221)

Note

If the DataWriter does not match with any DataReader that enables application acknowledgments (that is, a DataReader with **DDS_ReliabilityQosPolicy::acknowledgment_kind** (p. 1939) set to a value other than **DDS_←
_PROTOCOL_ACKNOWLEDGMENT_MODE** (p. 1323)), the value of *is_app_ack* is set to **DDS_BOOLEAN_←
_FALSE** (p. 1201).

5.17.4.78 DDS_DataWriter_wait_for_asynchronous_publishing()

```
DDS_ReturnCode_t DDS_DataWriter_wait_for_asynchronous_publishing (
    DDS_DataWriter * self,
    const struct DDS_Duration_t * max_wait )
```

<<**extension**>> (p. 1012) 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 **DDS_DataWriter** (p. 485) is sent and acknowledged (if reliable) by all matched **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615) is complete in addition to what **DDS_DataWriter_wait_for_acknowledgments** (p. 543) provides.

If the **DDS_DataWriter** (p. 485) does not have **DDS_PublishModeQosPolicy** (p. 1920) kind set to **DDS_↵ ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 1319) the operation will complete immediately , with **DDS_↵ RETCODE_OK** (p. 1222).

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>max_wait</i>	<< in >> (p. 1013) Specifies maximum time to wait for acknowledgements.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_NOT_ENABLED** (p. 1222), **DDS_RETCODE_↵ _TIMEOUT** (p. 1222)

5.17.4.79 DDS_DataWriter_get_liveliness_lost_status()

```
DDS_ReturnCode_t DDS_DataWriter_get_liveliness_lost_status (
    DDS_DataWriter * self,
    struct DDS_LivelinessLostStatus * status )
```

Accesses the **DDS_LIVELINESS_LOST_STATUS** (p. 1231) communication status.

This also resets the status so that it is no longer considered changed.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>status</i>	<< inout >> (p. 1013) DDS_LivelinessLostStatus (p. 1822) to be filled in. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.17.4.80 DDS_DataWriter_get_offered_deadline_missed_status()

```
DDS_ReturnCode_t DDS_DataWriter_get_offered_deadline_missed_status (
    DDS_DataWriter * self,
    struct DDS_OfferedDeadlineMissedStatus * status )
```

Accesses the **DDS_OFFERED_DEADLINE_MISSED_STATUS** (p. 1228) communication status.

This also resets the status so that it is no longer considered changed.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>status</i>	<< <i>inout</i> >> (p. 1013) DDS_OfferedDeadlineMissedStatus (p. 1861) to be filled in. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.17.4.81 DDS_DataWriter_get_offered_incompatible_qos_status()

```
DDS_ReturnCode_t DDS_DataWriter_get_offered_incompatible_qos_status (
    DDS_DataWriter * self,
    struct DDS_OfferedIncompatibleQosStatus * status )
```

Accesses the **DDS_OFFERED_INCOMPATIBLE_QOS_STATUS** (p. 1229) communication status.

This also resets the status so that it is no longer considered changed.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>status</i>	<< <i>inout</i> >> (p. 1013) DDS_OfferedIncompatibleQosStatus (p. 1862) to be filled in. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.17.4.82 DDS_DataWriter_get_publication_matched_status()

```
DDS_ReturnCode_t DDS_DataWriter_get_publication_matched_status (
    DDS_DataWriter * self,
    struct DDS_PublicationMatchedStatus * status )
```

Accesses the **DDS_PUBLICATION_MATCHED_STATUS** (p. 1232) communication status.

This also resets the status so that it is no longer considered changed.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>status</i>	<< <i>inout</i> >> (p. 1013) DDS_PublicationMatchedStatus (p. 1914) to be filled in. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.17.4.83 DDS_DataWriter_get_reliable_writer_cache_changed_status()

```
DDS_ReturnCode_t DDS_DataWriter_get_reliable_writer_cache_changed_status (
    DDS_DataWriter * self,
    struct DDS_ReliableWriterCacheChangedStatus * status )
```

<<*extension*>> (p. 1012) Get the reliable cache status for this writer.

This also resets the status so that it is no longer considered changed.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>status</i>	<< <i>inout</i> >> (p. 1013) DDS_ReliableWriterCacheChangedStatus (p. 1942) to be filled in. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.17.4.84 DDS_DataWriter_get_reliable_reader_activity_changed_status()

```
DDS_ReturnCode_t DDS_DataWriter_get_reliable_reader_activity_changed_status (
    DDS_DataWriter * self,
    struct DDS_ReliableReaderActivityChangedStatus * status )
```

<<**extension**>> (p. 1012) Get the reliable reader activity changed status for this writer.

This also resets the status so that it is no longer considered changed.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>status</i>	<< inout >> (p. 1013) DDS_ReliableReaderActivityChangedStatus (p. 1940) to be filled in. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.17.4.85 DDS_DataWriter_get_datawriter_cache_status()

```
DDS_ReturnCode_t DDS_DataWriter_get_datawriter_cache_status (
    DDS_DataWriter * self,
    struct DDS_DataWriterCacheStatus * status )
```

<<**extension**>> (p. 1012) Get the datawriter cache status for this writer.

This also resets the status so that it is no longer considered changed.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>status</i>	<< inout >> (p. 1013) DDS_DataWriterCacheStatus (p. 1641) to be filled in. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

5.17.4.86 DDS_DataWriter_get_datawriter_protocol_status()

```
DDS_ReturnCode_t DDS_DataWriter_get_datawriter_protocol_status (
    DDS_DataWriter * self,
    struct DDS_DataWriterProtocolStatus * status )
```

<<**extension**>> (p. 1012) Get the datawriter protocol status for this writer.

This also resets the status so that it is no longer considered changed.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>status</i>	<< <i>inout</i> >> (p. 1013) DDS_DataWriterProtocolStatus (p. 1654) to be filled in. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

5.17.4.87 **DDS_DataWriter_get_matched_subscription_datawriter_protocol_status()**

```
DDS_ReturnCode_t DDS_DataWriter_get_matched_subscription_datawriter_protocol_status (
    DDS_DataWriter * self,
    struct DDS_DataWriterProtocolStatus * status,
    const DDS_InstanceHandle_t * subscription_handle )
```

<<**extension**>> (p. 1012) 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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>status</i>	<< <i>inout</i> >> (p. 1013) DDS_DataWriterProtocolStatus (p. 1654) to be filled in. Cannot be NULL.
<i>subscription_handle</i>	<< <i>in</i> >> (p. 1013) Handle to a specific subscription associated with the DDS_DataReader (p. 615). Cannot be NULL. Must correspond to a subscription currently associated with the DDS_DataWriter (p. 485).

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

5.17.4.88 **DDS_DataWriter_get_matched_subscription_datawriter_protocol_status_by_locator()**

```
DDS_ReturnCode_t DDS_DataWriter_get_matched_subscription_datawriter_protocol_status_by_locator (
    DDS_DataWriter * self,
```

```
struct DDS_DataWriterProtocolStatus * status,
const struct DDS_Locator_t * locator )
```

<<**extension**>> (p. 1012) 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

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>status</i>	<< inout >> (p. 1013) DDS_DataWriterProtocolStatus (p. 1654) to be filled in Cannot be NULL.
<i>locator</i>	<< in >> (p. 1013) Locator to a specific locator associated with the DDS_DataReader (p. 615). Cannot be NULL. Must correspond to a locator of one or more subscriptions currently associated with the DDS_DataWriter (p. 485).

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

5.17.4.89 DDS_DataWriter_get_service_request_accepted_status()

```
DDS_ReturnCode_t DDS_DataWriter_get_service_request_accepted_status (
    DDS_DataWriter * self,
    struct DDS_ServiceRequestAcceptedStatus * status )
```

Accesses the **DDS_SERVICE_REQUEST_ACCEPTED_STATUS** (p. 1233) communication status.

This also resets the status so that it is no longer considered changed.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>status</i>	<< inout >> (p. 1013) DDS_ServiceRequestAcceptedStatus (p. 2002) to be filled in. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.17.4.90 DDS_DataWriter_set_qos()

```
DDS_ReturnCode_t DDS_DataWriter_set_qos (
    DDS_DataWriter * self,
    const struct DDS_DataWriterQos * qos )
```

Sets the writer QoS.

This operation modifies the QoS of the **DDS_DataWriter** (p. 485).

The **DDS_DataWriterQos::user_data** (p. 1670), **DDS_DataWriterQos::deadline** (p. 1668), **DDS_DataWriterQos::latency_budget** (p. 1668), **DDS_DataWriterQos::ownership_strength** (p. 1670), **DDS_DataWriterQos::transport_priority** (p. 1669), **DDS_DataWriterQos::lifespan** (p. 1669) and **DDS_DataWriterQos::writer_data_lifecycle** (p. 1670) can be changed. The other policies are immutable.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>in</i> >> (p. 1013) The DDS_DataWriterQos (p. 1666) to be set to. Policies must be consistent. Immutable policies cannot be changed after DDS_DataWriter (p. 485) is enabled. The special value DDS_DATAWRITER_QOS_DEFAULT (p. 470) can be used to indicate that the QoS of the DDS_DataWriter (p. 485) should be changed to match the current default DDS_DataWriterQos (p. 1666) set in the DDS_Publisher (p. 443). Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 1222) or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222)

See also

DDS_DataWriterQos (p. 1666) for rules on consistency among QoS
set_qos (abstract) (p. 1361)
Operations Allowed in Listener Callbacks (p. 1818)

5.17.4.91 DDS_DataWriter_set_qos_with_profile()

```
DDS_ReturnCode_t DDS_DataWriter_set_qos_with_profile (
    DDS_DataWriter * self,
    const char * library_name,
    const char * profile_name )
```

<<*extension*>> (p. 1012) Change the QoS of this writer using the input XML QoS profile.

This operation modifies the QoS of the **DDS_DataWriter** (p. 485).

The **DDS_DataWriterQos::user_data** (p. 1670), **DDS_DataWriterQos::deadline** (p. 1668), **DDS_DataWriterQos::latency_budget** (p. 1668), **DDS_DataWriterQos::ownership_strength** (p. 1670), **DDS_DataWriterQos::transport_priority** (p. 1669), **DDS_DataWriterQos::lifespan** (p. 1669) and **DDS_DataWriterQos::writer_data_lifecycle** (p. 1670) can be changed. The other policies are immutable.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< <i>in</i> >> (p. 1013) Library name containing the XML QoS profile. If <i>library_name</i> is null RTI Connex will use the default library (see DDS_Publisher_set_default_library (p. 452)).
<i>profile_name</i>	<< <i>in</i> >> (p. 1013) XML QoS Profile name. If <i>profile_name</i> is null RTI Connex will use the default profile (see DDS_Publisher_set_default_profile (p. 451)).

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 1222) or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222)

See also

DDS_DataWriterQos (p. 1666) for rules on consistency among QoS
Operations Allowed in Listener Callbacks (p. 1818)

5.17.4.92 DDS_DataWriter_get_qos()

```
DDS_ReturnCode_t DDS_DataWriter_get_qos (
    DDS_DataWriter * self,
    struct DDS_DataWriterQos * qos )
```

Gets the writer QoS.

This function may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>inout</i> >> (p. 1013) The DDS_DataWriterQos (p. 1666) to be filled up. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

get_qos (abstract) (p. 1362)

5.17.4.93 DDS_DataWriter_set_property()

```
DDS_ReturnCode_t DDS_DataWriter_set_property (
    DDS_DataWriter * self,
    const char * property_name,
    const char * value,
    DDS_Boolean propagate )
```

Set the value for a property that applies to a DataWriter.

Warning

This function is not implemented in all APIs and it's intended only for testing purposes. You should use **DDS_↔DataWriter_set_qos** (p. 550) instead.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>property_name</i>	<< <i>in</i> >> (p. 1013). Name of the property that you want to set.
<i>value</i>	<< <i>in</i> >> (p. 1013). New value for the property.
<i>propagate</i>	<< <i>in</i> >> (p. 1013). Indicates if the property will be propagated or not.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DomainParticipant_set_property (p. 153)

DDS_DataReader_set_property (p. 689)

DDS_DataWriter_set_qos (p. 550)

5.17.4.94 DDS_DataWriter_set_listener()

```
DDS_ReturnCode_t DDS_DataWriter_set_listener (
    DDS_DataWriter * self,
    const struct DDS_DataWriterListener * l,
    DDS_StatusMask mask )
```

Sets the writer listener.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>l</i>	<< <i>in</i> >> (p. 1013) DDS_DataWriterListener (p. 1644) to set to
<i>mask</i>	<< <i>in</i> >> (p. 1013) DDS_StatusMask (p. 1226) associated with the DDS_DataWriterListener (p. 1644). The callback function on the listener cannot be NULL if the corresponding status is turned on in the <i>mask</i> .

Returns

One of the **Standard Return Codes** (p. 1221)

See also

set_listener (abstract) (p. 1362)

5.17.4.95 DDS_DataWriter_get_listener()

```
struct DDS_DataWriterListener DDS_DataWriter_get_listener (
    DDS_DataWriter * self )
```

Get the writer listener.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_DataWriterListener (p. 1644) of the **DDS_DataWriter** (p. 485).

See also

DDS_DataWriter_get_listenerX (p. 554)

get_listener (abstract) (p. 1363)

5.17.4.96 DDS_DataWriter_get_listenerX()

```
DDS_ReturnCode_t DDS_DataWriter_get_listenerX (
    DDS_DataWriter * self,
    struct DDS_DataWriterListener * listener )
```

<<*extension*>> (p. 1012) Get the writer listener.

An alternative form of `get_listener` that fills in an existing listener structure rather than returning one on the stack.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>listener</i>	<< <i>inout</i> >> (p. 1013) DDS_DataWriterListener (p. 1644) structure to be filled up. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataWriter_get_listener (p. 554)

get_listener (abstract) (p. 1363)

5.17.4.97 DDS_DataWriter_flush()

```
DDS_ReturnCode_t DDS_DataWriter_flush (
    DDS_DataWriter * self )
```

<<**extension**>> (p. 1012) 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 **DDS_DataWriter** (p. 485) does not have **DDS_PublishModeQosPolicy** (p. 1920) kind set to **DDS_↔ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 1319), the batch will be sent on the network immediately (in the context of the calling thread).

If the **DDS_DataWriter** (p. 485) does have **DDS_PublishModeQosPolicy** (p. 1920) kind set to **DDS_↔ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 1319), 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. 496).

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() is only thread-safe with batching if **DDS_BatchQosPolicy::thread_safe_write** (p. 1555) is **TRUE**.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_TIMEOUT** (p. 1222), **DDS_RETCODE_OUT_↔OF_RESOURCES** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

5.17.4.98 DDS_DataWriter_take_discovery_snapshot()

```
DDS_ReturnCode_t DDS_DataWriter_take_discovery_snapshot (
    DDS_DataWriter * self,
    const char * file_name )
```

Take a snapshot of the compatible and incompatible remote readers matched by a local writer.

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="udp4://192.168.1.170:7411"

liveliness="ALIVE"

Incompatible readers:

1. 0x0101542A,0x2C59B595,0xA1693BDF:0x80000104 name="reader2TestName"

kind="unkeyed user datareader"

unicastLocators="udp4://192.168.1.170:7411"

reason="Inconsistent QoS"

Precondition

self is not NULL.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) The local writer.
<i>file_name</i>	<< <i>in</i> >> (p. 1013) Name of the file where snapshot should be printed. If NULL, the snapshot will be printed through NDDS_Config_Logger (p. 2116). Otherwise, the snapshot will be printed in the file specified by <i>file_name</i> .

Returns

One of the **Standard Return Codes** (p. 1221).

5.18 Flow Controllers

<<*extension*>> (p. 1012) **DDS_FlowController** (p. 559) and associated elements

Data Structures

- struct **DDS_FlowControllerTokenBucketProperty_t**
DDS_FlowController (p. 559) 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 DDS_FlowController (p. 559).

Macros

- #define **DDS_FlowControllerTokenBucketProperty_t_INITIALIZER**
Initializer for new property instances.

Typedefs

- typedef struct DDS_FlowControllerImpl **DDS_FlowController**
<<interface>> (p. 1013) A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous DDS_DataWriter (p. 485) 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.

Functions

- const char * **DDS_FlowController_get_name** (DDS_FlowController *self)
Returns the name of the DDS_FlowController (p. 559).
- DDS_DomainParticipant * **DDS_FlowController_get_participant** (DDS_FlowController *self)
Returns the DDS_DomainParticipant (p. 77) *to which the DDS_FlowController* (p. 559) *belongs.*
- DDS_ReturnCode_t **DDS_FlowController_set_property** (DDS_FlowController *self, const struct **DDS_FlowControllerProperty_t** *prop)
Sets the DDS_FlowController (p. 559) *property.*
- DDS_ReturnCode_t **DDS_FlowController_get_property** (DDS_FlowController *self, struct **DDS_FlowControllerProperty_t** *prop)
Gets the DDS_FlowController (p. 559) *property.*
- DDS_ReturnCode_t **DDS_FlowController_trigger_flow** (DDS_FlowController *self)
Provides an external trigger to the DDS_FlowController (p. 559).

Variables

- char * **DDS_DEFAULT_FLOW_CONTROLLER_NAME**
[default] Special value of **DDS_PublishModeQosPolicy::flow_controller_name** (p. 1922) 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. 1922) 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. 1922) that refers to the built-in on-demand flow controller.

5.18.1 Detailed Description

<<*extension*>> (p. 1012) **DDS_FlowController** (p. 559) and associated elements

DDS_FlowController (p. 559) provides the network traffic shaping capability to asynchronous **DDS_DataWriter** (p. 485) instances. For use cases and advantages of publishing asynchronously, please refer to **DDS_PublishModeQosPolicy** (p. 1920) of **DDS_DataWriterQos** (p. 1666).

See also

DDS_PublishModeQosPolicy (p. 1920)
DDS_DataWriterQos::publish_mode (p. 1671)
DDS_AsynchronousPublisherQosPolicy (p. 1539)

5.18.2 Macro Definition Documentation

5.18.2.1 DDS_FlowControllerTokenBucketProperty_t_INITIALIZER

```
#define DDS_FlowControllerTokenBucketProperty_t_INITIALIZER
```

Initializer for new property instances.

New **DDS_FlowControllerProperty_t** (p. 1799) instances stored on the stack should be initialized with this value before they are passed to any function. This step ensures that the contained property values are properly initialized. This does not allocate memory.

The simplest way to create a new property structure is to initialize it on the stack at the time of its creation:

```
struct DDS_FlowControllerProperty_t myProperty = DDS_FlowControllerProperty_t_INITIALIZER;
```

Note that the above assignment is not a substitute for calling **DDS_DomainParticipant_get_default_flowcontroller_property** (p. 98).

See also

DDS_DomainParticipant_get_default_flowcontroller_property (p. 98)

5.18.3 Typedef Documentation

5.18.3.1 DDS_FlowController

```
typedef struct DDS_FlowControllerImpl DDS_FlowController
```

<<*interface*>> (p. 1013) A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous **DDS_DataWriter** (p. 485) instances are allowed to write data.

QoS:

DDS_FlowControllerProperty_t (p. 1799)

5.18.4 Enumeration Type Documentation

5.18.4.1 DDS_FlowControllerSchedulingPolicy

```
enum DDS_FlowControllerSchedulingPolicy
```

Kinds of flow controller scheduling policy.

Samples written by an asynchronous **DDS_DataWriter** (p. 485) are not sent in the context of the **FooDataWriter_write** (p. 496) call. Instead, the middleware puts the samples in a queue for future processing. The **DDS_FlowController** (p. 559) associated with each asynchronous DataWriter instance determines when the samples are actually sent.

Each **DDS_FlowController** (p. 559) maintains a separate FIFO queue for each unique destination (remote application). Samples written by asynchronous **DDS_DataWriter** (p. 485) 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. 1799)

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 <code>DataWriter</code> at the time of the write call (as specified in the DDS_LatencyBudgetQosPolicy (p. 1813)). 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 DDS_FlowController (p. 559) 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. 1815) setting, an <code>EDF_FLOW_CONTROLLER_SCHED_POLICY</code> DDS_FlowController (p. 559) preserves the order in which the user calls FooDataWriter_write (p. 496) across the <code>DataWriters</code> associated with the flow controller. Since the DDS_LatencyBudgetQosPolicy (p. 1813) is mutable, a sample written second may contain an earlier deadline than the sample written first if the DDS_LatencyBudgetQosPolicy::duration (p. 1815) 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. 1815) and adhere to the DDS_DataWriter (p. 485) in-order data delivery guarantee. [default] for DDS_DataWriter (p. 485)

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 DDS_DataWriter (p. 485), 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 DDS_FlowController (p. 559) 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 DDS_DataWriter (p. 485) in-order data delivery guarantee.

5.18.5 Function Documentation

5.18.5.1 DDS_FlowController_get_name()

```
const char * DDS_FlowController_get_name (
    DDS_FlowController * self )
```

Returns the name of the **DDS_FlowController** (p. 559).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

The name of the **DDS_FlowController** (p. 559).

5.18.5.2 DDS_FlowController_get_participant()

```
DDS_DomainParticipant * DDS_FlowController_get_participant (
    DDS_FlowController * self )
```

Returns the **DDS_DomainParticipant** (p. 77) to which the **DDS_FlowController** (p. 559) belongs.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

The **DDS_DomainParticipant** (p. 77) to which the **DDS_FlowController** (p. 559) belongs.

5.18.5.3 DDS_FlowController_set_property()

```
DDS_ReturnCode_t DDS_FlowController_set_property (
    DDS_FlowController * self,
    const struct DDS_FlowControllerProperty_t * prop )
```

Sets the **DDS_FlowController** (p. 559) property.

This operation modifies the property of the **DDS_FlowController** (p. 559).

Once a **DDS_FlowController** (p. 559) has been instantiated, only the **DDS_FlowControllerProperty_t::token_bucket** (p. 1800) can be changed. The **DDS_FlowControllerProperty_t::scheduling_policy** (p. 1799) is immutable.

A new **DDS_FlowControllerTokenBucketProperty_t::period** (p. 1802) only takes effect at the next scheduled token distribution time (as determined by its previous value).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>prop</i>	<< <i>in</i> >> (p. 1013) The new DDS_FlowControllerProperty_t (p. 1799). Property must be consistent. Immutable fields cannot be changed after DDS_FlowController (p. 559) has been created. The special value DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT (p. 162) can be used to indicate that the property of the DDS_FlowController (p. 559) should be changed to match the current default DDS_FlowControllerProperty_t (p. 1799) set in the DDS_DomainParticipant (p. 77). Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 1222), or **DDS_↵
RETCODE_INCONSISTENT_POLICY** (p. 1222).

See also

DDS_FlowControllerProperty_t (p. 1799) for rules on consistency among property values.

5.18.5.4 DDS_FlowController_get_property()

```
DDS_ReturnCode_t DDS_FlowController_get_property (
    DDS_FlowController * self,
    struct DDS_FlowControllerProperty_t * prop )
```

Gets the **DDS_FlowController** (p. 559) property.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>prop</i>	<< <i>in</i> >> (p. 1013) DDS_FlowController (p. 559) to be filled in. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.18.5.5 DDS_FlowController_trigger_flow()

```
DDS_ReturnCode_t DDS_FlowController_trigger_flow (
    DDS_FlowController * self )
```

Provides an external trigger to the **DDS_FlowController** (p. 559).

Typically, a **DDS_FlowController** (p. 559) 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. 1802) property setting.

This function provides an additional, external trigger to the **DDS_FlowController** (p. 559). This trigger adds **DDS_FlowControllerTokenBucketProperty_t::tokens_added_per_period** (p. 1801) tokens each time it is called (subject to the other property settings of the **DDS_FlowController** (p. 559)).

An *on-demand* **DDS_FlowController** (p. 559) can be created with a **DDS_DURATION_INFINITE** (p. 1208) as **DDS_FlowControllerTokenBucketProperty_t::period** (p. 1802), in which case the only trigger source is external (i.e. the FlowController is solely triggered by the user on demand).

DDS_FlowController_trigger_flow (p. 563) can be called on both strict *on-demand* FlowController and hybrid FlowController (internally and externally triggered).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.18.6 Variable Documentation

5.18.6.1 DDS_DEFAULT_FLOW_CONTROLLER_NAME

```
char* DDS_DEFAULT_FLOW_CONTROLLER_NAME [extern]
```

[default] Special value of **DDS_PublishModeQosPolicy::flow_controller_name** (p. 1922) that refers to the built-in default flow controller.

RTI Connext provides several built-in **DDS_FlowController** (p. 559) for use with an asynchronous **DDS_DataWriter** (p. 485). 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. 564) flow controller does not apply any flow control. Instead, it allows data to be sent asynchronously as soon as it is written by the **DDS_DataWriter** (p. 485).

Essentially, this is equivalent to a user-created **DDS_FlowController** (p. 559) with the following **DDS_FlowControllerProperty_t** (p. 1799) settings:

- **DDS_FlowControllerProperty_t::scheduling_policy** (p. 1799) = **DDS_EDF_FLOW_CONTROLLER_SCHED_POLICY** (p. 560)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1800) `max_tokens` = **DDS_LENGTH_UNLIMITED** (p. 1326)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1800) `tokens_added_per_period` = **DDS_LENGTH_UNLIMITED** (p. 1326)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1800) `tokens_leaked_per_period` = 0
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1800) `period` = 60 seconds
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1800) `bytes_per_token` = **DDS_LENGTH_UNLIMITED** (p. 1326)

See also

DDS_Publisher_create_datawriter (p. 453)
DDS_DomainParticipant_lookup_flowcontroller (p. 128)
DDS_FlowController_set_property (p. 562)
DDS_PublishModeQosPolicy (p. 1920)
DDS_AsynchronousPublisherQosPolicy (p. 1539)

5.18.6.2 DDS_FIXED_RATE_FLOW_CONTROLLER_NAME

```
char* DDS_FIXED_RATE_FLOW_CONTROLLER_NAME [extern]
```

Special value of **DDS_PublishModeQosPolicy::flow_controller_name** (p. 1922) that refers to the built-in fixed-rate flow controller.

RTI Connext provides several builtin **DDS_FlowController** (p. 559) for use with an asynchronous **DDS_DataWriter** (p. 485). 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. 564) 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 **DDS_FlowController** (p. 559) with the following **DDS_FlowControllerProperty_t** (p. 1799) settings:

- **DDS_FlowControllerProperty_t::scheduling_policy** (p. 1799) = **DDS_EDF_FLOW_CONTROLLER_SCHED_POLICY** (p. 560)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1800) `max_tokens` = **DDS_LENGTH_UNLIMITED** (p. 1326)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1800) `tokens_added_per_period` = **DDS_LENGTH_UNLIMITED** (p. 1326)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1800) `tokens_leaked_per_period` = **DDS_LENGTH_UNLIMITED** (p. 1326)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1800) `period` = 1 second
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1800) `bytes_per_token` = **DDS_LENGTH_UNLIMITED** (p. 1326)

See also

DDS_Publisher_create_datawriter (p. 453)
DDS_DomainParticipant_lookup_flowcontroller (p. 128)
DDS_FlowController_set_property (p. 562)
DDS_PublishModeQosPolicy (p. 1920)
DDS_AsynchronousPublisherQosPolicy (p. 1539)

5.18.6.3 DDS_ON_DEMAND_FLOW_CONTROLLER_NAME

```
char* DDS_ON_DEMAND_FLOW_CONTROLLER_NAME [extern]
```

Special value of **DDS_PublishModeQosPolicy::flow_controller_name** (p. 1922) that refers to the built-in on-demand flow controller.

RTI Connext provides several builtin **DDS_FlowController** (p. 559) for use with an asynchronous **DDS_DataWriter** (p. 485). 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. 565) allows data to be sent only when the user calls **DDS_FlowController_trigger_flow** (p. 563). With each trigger, all accumulated data since the previous trigger is sent (across all **DDS_Publisher** (p. 443) or **DDS_DataWriter** (p. 485) 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. 2124)).

Essentially, this is equivalent to a user-created **DDS_FlowController** (p. 559) with the following **DDS_FlowController_Property_t** (p. 1799) settings:

- **DDS_FlowControllerProperty_t::scheduling_policy** (p. 1799) = **DDS_EDF_FLOW_CONTROLLER_SCHED_POLICY** (p. 560)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1800) `max_tokens` = **DDS_LENGTH_UNLIMITED** (p. 1326)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1800) `tokens_added_per_period` = **DDS_LENGTH_UNLIMITED** (p. 1326)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1800) `tokens_leaked_per_period` = **DDS_LENGTH_UNLIMITED** (p. 1326)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1800) `period` = **DDS_DURATION_INFINITE** (p. 1208)
- **DDS_FlowControllerProperty_t::token_bucket** (p. 1800) `bytes_per_token` = **DDS_LENGTH_UNLIMITED** (p. 1326)

See also

DDS_Publisher_create_datawriter (p. 453)
DDS_DomainParticipant_lookup_flowcontroller (p. 128)
DDS_FlowController_trigger_flow (p. 563)
DDS_FlowController_set_property (p. 562)
DDS_PublishModeQosPolicy (p. 1920)
DDS_AsynchronousPublisherQosPolicy (p. 1539)

5.19 Subscription Module

Contains the **DDS_Subscriber** (p. 572), **DDS_DataReader** (p. 615), **DDS_ReadCondition** (p. 694), **DDS_QueryCondition** (p. 699), and **DDS_TopicQuery** (p. 707) classes, as well as the **DDS_SubscriberListener** (p. 2009) and **DDS_DataReaderListener** (p. 1593) interfaces, and more generally, all that is needed on the subscription side.

Modules

- **Subscribers**
 - DDS_Subscriber* (p. 572) entity and associated elements
- **DataReaders**
 - DDS_DataReader* (p. 615) entity and associated elements
- **Data Samples**
 - DDS_SampleInfo* (p. 1985), *DDS_SampleStateKind* (p. 710), *DDS_ViewStateKind* (p. 712), *DDS_InstanceStateKind* (p. 714) and associated elements
- **SampleProcessor**
 - <<experimental>>* (p. 1012) *<<extension>>* (p. 1012) Utility to concurrently read and process the data samples received by *DDS_DataReader* (p. 615).

5.19.1 Detailed Description

Contains the **DDS_Subscriber** (p. 572), **DDS_DataReader** (p. 615), **DDS_ReadCondition** (p. 694), **DDS_QueryCondition** (p. 699), and **DDS_TopicQuery** (p. 707) classes, as well as the **DDS_SubscriberListener** (p. 2009) and **DDS_DataReaderListener** (p. 1593) interfaces, and more generally, all that is needed on the subscription side.

"DCPS Subscription package"

5.19.2 Access to data samples

Data is made available to the application by the following operations on **DDS_DataReader** (p. 615) objects: **FooDataReader_read** (p. 625), **FooDataReader_read_w_condition** (p. 630), **FooDataReader_take** (p. 626), **FooDataReader_take_w_condition** (p. 632), 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. 1985) 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. 1292) QoS allow for it.

These operations reset the read communication statuses; see **Changes in read communication status** (p. 1225).

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. 646).

See also

Interpretation of the `SampleInfo` (p. 1986)

5.19.2.1 Data access patterns

The application accesses data by means of the operations `read` or `take` on the **DDS_DataReader** (p. 615). These operations return an ordered collection of `DataSamples` consisting of a **DDS_SampleInfo** (p. 1985) part and a `Data` part.

The way RTI Connex builds the collection depends on QoS policies set on the **DDS_DataReader** (p. 615) and **DDS_Subscriber** (p. 572), 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. 710))
- the desired view states (any combination of **DDS_ViewStateKind** (p. 712))
- the desired instance states (any combination of **DDS_InstanceStateKind** (p. 714))

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 **DDS_ReadCondition** (p. 694) 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. 1201). These operations, in conjunction with **DDS_ReadCondition** (p. 694) objects and a **DDS_WaitSet** (p. 1371), 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. 1304) QoS must be set properly.

5.20 Subscribers

DDS_Subscriber (p. 572) entity and associated elements

Data Structures

- struct **DDS_SubscriberSeq**
Declares IDL sequence < DDS_Subscriber (p. 572) > .
- struct **DDS_SubscriberQos**
QoS policies supported by a DDS_Subscriber (p. 572) entity.
- struct **DDS_SubscriberListener**
<<interface>> (p. 1013) DDS_Listener (p. 1816) for status about a subscriber.

Macros

- #define **DDS_SubscriberQos_INITIALIZER**
Initializer for new QoS instances.
- #define **DDS_SubscriberListener_INITIALIZER**
<<extension>> (p. 1012) Initializer for new DDS_SubscriberListener (p. 2009).

Typedefs

- typedef struct DDS_SubscriberImpl **DDS_Subscriber**
<<interface>> (p. 1013) A subscriber is the object responsible for actually receiving data from a subscription.
- typedef void(* **DDS_SubscriberListener_DataOnReadersCallback**) (void *listener_data, **DDS_Subscriber** *sub)
*Prototype of a **DDS_SubscriberListener** (p. 2009) data_on_readers function.*

Functions

- **DDS_Boolean DDS_SubscriberQos_equals** (const struct **DDS_SubscriberQos** *self, const struct **DDS_SubscriberQos** *other)
*Compares two **DDS_SubscriberQos** (p. 2010) for equality.*
- **DDS_ReturnCode_t DDS_SubscriberQos_print** (const struct **DDS_SubscriberQos** *self)
*Prints this **DDS_SubscriberQos** (p. 2010) to stdout.*
- **DDS_ReturnCode_t DDS_SubscriberQos_to_string** (const struct **DDS_SubscriberQos** *self, char *string, **DDS_UnsignedLong** *string_size)
*Obtains a string representation of this **DDS_SubscriberQos** (p. 2010).*
- **DDS_ReturnCode_t DDS_SubscriberQos_to_string_w_params** (const struct **DDS_SubscriberQos** *self, char *string, **DDS_UnsignedLong** *string_size, const struct **DDS_SubscriberQos** *base, const struct **DDS_QosPrintFormat** *format)
*Obtains a string representation of this **DDS_SubscriberQos** (p. 2010).*
- **DDS_ReturnCode_t DDS_SubscriberQos_initialize** (struct **DDS_SubscriberQos** *self)
Initializer for new QoS instances.
- **DDS_ReturnCode_t DDS_SubscriberQos_finalize** (struct **DDS_SubscriberQos** *self)
*Free any dynamic memory allocated by the policies in this **DDS_SubscriberQos** (p. 2010).*
- **DDS_ReturnCode_t DDS_SubscriberQos_copy** (struct **DDS_SubscriberQos** *self, const struct **DDS_SubscriberQos** *source)
Copy the contents of the given QoS into this QoS.
- **DDS_Entity * DDS_Subscriber_as_entity** (**DDS_Subscriber** *subscriber)
*Access a **DDS_Subscriber** (p. 572)'s supertype instance.*
- **DDS_ReturnCode_t DDS_Subscriber_get_default_datareader_qos** (**DDS_Subscriber** *self, struct **DDS_DataReaderQos** *qos)
*Copies the default **DDS_DataReaderQos** (p. 1611) values into the provided **DDS_DataReaderQos** (p. 1611) instance.*
- **DDS_ReturnCode_t DDS_Subscriber_set_default_datareader_qos** (**DDS_Subscriber** *self, const struct **DDS_DataReaderQos** *qos)
*Sets the default **DDS_DataReaderQos** (p. 1611) values for this subscriber.*
- **DDS_ReturnCode_t DDS_Subscriber_set_default_datareader_qos_with_profile** (**DDS_Subscriber** *self, const char *library_name, const char *profile_name)
*<<extension>> (p. 1012) Set the default **DDS_DataReaderQos** (p. 1611) values for this subscriber based on the input XML QoS profile.*
- **DDS_DataReader * DDS_Subscriber_create_datareader** (**DDS_Subscriber** *self, **DDS_TopicDescription** *topic, const struct **DDS_DataReaderQos** *qos, const struct **DDS_DataReaderListener** *listener, **DDS_StatusMask** mask)
*Creates a **DDS_DataReader** (p. 615) that will be attached and belong to the **DDS_Subscriber** (p. 572).*
- **DDS_DataReader * DDS_Subscriber_create_datareader_with_profile** (**DDS_Subscriber** *self, **DDS_TopicDescription** *topic, const char *library_name, const char *profile_name, const struct **DDS_DataReaderListener** *listener, **DDS_StatusMask** mask)

<<**extension**>> (p. 1012) Creates a **DDS_DataReader** (p. 615) object using the **DDS_DataReaderQos** (p. 1611) associated with the input XML QoS profile.

- **DDS_ReturnCode_t DDS_Subscriber_delete_datareader** (**DDS_Subscriber** *self, **DDS_DataReader** *a↵_datareader)
*Deletes a **DDS_DataReader** (p. 615) that belongs to the **DDS_Subscriber** (p. 572).*
- **DDS_ReturnCode_t DDS_Subscriber_delete_contained_entities** (**DDS_Subscriber** *self)
*Deletes all the entities that were created by means of the "create" operation on the **DDS_Subscriber** (p. 572).*
- **DDS_DataReader * DDS_Subscriber_lookup_datareader** (**DDS_Subscriber** *self, const char *topic_name)
*Retrieves an existing **DDS_DataReader** (p. 615).*
- **DDS_ReturnCode_t DDS_Subscriber_begin_access** (**DDS_Subscriber** *self)
*Indicates that the application is about to access the data samples in any of the **DDS_DataReader** (p. 615) objects attached to the **DDS_Subscriber** (p. 572).*
- **DDS_ReturnCode_t DDS_Subscriber_end_access** (**DDS_Subscriber** *self)
*Indicates that the application has finished accessing the data samples in **DDS_DataReader** (p. 615) objects managed by the **DDS_Subscriber** (p. 572).*
- **DDS_ReturnCode_t DDS_Subscriber_get_datareaders** (**DDS_Subscriber** *self, struct **DDS_DataReader**↵Seq *readers, **DDS_SampleStateMask** sample_states, **DDS_ViewStateMask** view_states, **DDS_Instance**↵StateMask instance_states)
*Allows the application to access the **DDS_DataReader** (p. 615) objects that contain samples with the specified **sample**↵_states, **view_states** and **instance_states**.*
- **DDS_ReturnCode_t DDS_Subscriber_get_all_datareaders** (**DDS_Subscriber** *self, struct **DDS_Data**↵ReaderSeq *readers)
Retrieve all the DataReaders created from this Subscriber.
- **DDS_ReturnCode_t DDS_Subscriber_notify_datareaders** (**DDS_Subscriber** *self)
*Invokes the operation **DDS_DataReaderListener::on_data_available()** (p. 1595) on the **DDS_DataReaderListener** (p. 1593) objects attached to contained **DDS_DataReader** (p. 615) entities with **DDS_DATA_AVAILABLE_STATUS** (p. 1230) that is considered changed as described in **Changes in read communication status** (p. 1225).*
- **DDS_DomainParticipant * DDS_Subscriber_get_participant** (**DDS_Subscriber** *self)
*Returns the **DDS_DomainParticipant** (p. 77) to which the **DDS_Subscriber** (p. 572) belongs.*
- **DDS_ReturnCode_t DDS_Subscriber_copy_from_topic_qos** (**DDS_Subscriber** *self, struct **DDS_Data**↵ReaderQos *datareader_qos, const struct **DDS_TopicQos** *topic_qos)
*Copies the policies in the **DDS_TopicQos** (p. 2042) to the corresponding policies in the **DDS_DataReaderQos** (p. 1611).*
- **DDS_ReturnCode_t DDS_Subscriber_set_qos** (**DDS_Subscriber** *self, const struct **DDS_SubscriberQos** *qos)
Sets the subscriber QoS.
- **DDS_ReturnCode_t DDS_Subscriber_set_qos_with_profile** (**DDS_Subscriber** *self, const char *library↵_name, const char *profile_name)
 <<**extension**>> (p. 1012) Change the QoS of this subscriber using the input XML QoS profile.
- **DDS_ReturnCode_t DDS_Subscriber_get_qos** (**DDS_Subscriber** *self, struct **DDS_SubscriberQos** *qos)
Gets the subscriber QoS.
- **DDS_ReturnCode_t DDS_Subscriber_set_default_profile** (**DDS_Subscriber** *self, const char *library↵_name, const char *profile_name)
 <<**extension**>> (p. 1012) Sets the default XML profile for a **DDS_Subscriber** (p. 572).
- const char * **DDS_Subscriber_get_default_profile** (**DDS_Subscriber** *self)
 <<**extension**>> (p. 1012) Gets the default XML profile associated with a **DDS_Subscriber** (p. 572).
- const char * **DDS_Subscriber_get_default_profile_library** (**DDS_Subscriber** *self)
 <<**extension**>> (p. 1012) Gets the library where the default XML QoS profile is contained for a **DDS_Subscriber** (p. 572).
- **DDS_ReturnCode_t DDS_Subscriber_set_default_library** (**DDS_Subscriber** *self, const char *library↵_name)

- <<*extension*>> (p. 1012) Sets the default XML library for a **DDS_Subscriber** (p. 572).
- const char * **DDS_Subscriber_get_default_library** (**DDS_Subscriber** *self)
 - <<*extension*>> (p. 1012) Gets the default XML library associated with a **DDS_Subscriber** (p. 572).
- **DDS_ReturnCode_t DDS_Subscriber_set_listener** (**DDS_Subscriber** *self, const struct **DDS_SubscriberListener** *l, **DDS_StatusMask** mask)
 - Sets the subscriber listener.
- struct **DDS_SubscriberListener DDS_Subscriber_get_listener** (**DDS_Subscriber** *self)
 - Get the subscriber listener.
- **DDS_ReturnCode_t DDS_Subscriber_get_listenerX** (**DDS_Subscriber** *self, struct **DDS_SubscriberListener** *listener)
 - <<*extension*>> (p. 1012) Get the subscriber listener.
- **DDS_DataReader * DDS_Subscriber_lookup_datareader_by_name** (**DDS_Subscriber** *self, const char *datareader_name)
 - <<*extension*>> (p. 1012) Retrieves a **DDS_DataReader** (p. 615) contained within the **DDS_Subscriber** (p. 572) the **DDS_DataReader** (p. 615) entity name.

Variables

- const struct **DDS_DataReaderQos * DDS_DATAREADER_QOS_PRINT_ALL**
 - Special value which can be supplied to **DDS_DataReaderQos_to_string_w_params** (p. 668) indicating that all of the QoS should be printed.
- 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 **DDS_DataReader** (p. 615) with a combination of the default **DDS_DataReaderQos** (p. 1611) and the **DDS_TopicQos** (p. 2042).

5.20.1 Detailed Description

DDS_Subscriber (p. 572) entity and associated elements

5.20.2 Macro Definition Documentation

5.20.2.1 DDS_SubscriberQos_INITIALIZER

```
#define DDS_SubscriberQos_INITIALIZER
```

Initializer for new QoS instances.

New **DDS_SubscriberQos** (p. 2010) instances stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those contained QoS policies that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a new QoS structure is to initialize it on the stack at the time of its creation:

```
struct DDS_SubscriberQos myQos = DDS_SubscriberQos_INITIALIZER;
```

Note that the above assignment is not a substitute for calling **DDS_DomainParticipant_get_default_subscriber_qos** (p. 94) or **DDS_Subscriber_get_qos** (p. 593); one of those should be called subsequently to setting the QoS of a new or existing entity. **DDS_SubscriberQos_finalize** (p. 577) should be called to free the contained QoS policies that use dynamic memory:

```
struct DDS_SubscriberQos myQos = DDS_SubscriberQos_INITIALIZER;

DDS_DomainParticipant_get_default_subscriber_qos(myParticipant, &myQos);

DDS_Subscriber_set_qos(mySubscriber, &myQos);

DDS_SubscriberQos_finalize(&myQos);
```

See also

DDS_SubscriberQos_initialize (p. 577)

DDS_DomainParticipant_get_default_subscriber_qos (p. 94)

DDS_SubscriberQos_finalize (p. 577)

5.20.2.2 DDS_SubscriberListener_INITIALIZER

```
#define DDS_SubscriberListener_INITIALIZER
```

<<**extension**>> (p. 1012) Initializer for new **DDS_SubscriberListener** (p. 2009).

All the new instances allocated in the stack should be initialized to this value. No memory is allocated.

5.20.3 Typedef Documentation

5.20.3.1 DDS_Subscriber

```
typedef struct DDS_SubscriberImpl DDS_Subscriber
```

<<**interface**>> (p. 1013) A subscriber is the object responsible for actually receiving data from a subscription.

QoS:

DDS_SubscriberQos (p. 2010)

Status:

DDS_DATA_ON_READERS_STATUS (p. 1230)

Listener:

DDS_SubscriberListener (p. 2009)

A subscriber acts on the behalf of one or several **DDS_DataReader** (p. 615) objects that are related to it. When it receives data (from the other parts of the system), it builds the list of concerned **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615) objects through the operation **DDS_Subscriber_get_datareaders** (p. 588) and then access the data available through operations on the **DDS_DataReader** (p. 615).

The following operations may be called even if the **DDS_Subscriber** (p. 572) is not enabled. Other operations will the value **DDS_RETCODE_NOT_ENABLED** (p. 1222) if called on a disabled **DDS_Subscriber** (p. 572):

- **DDS_Entity_enable** (p. 1363),
- **DDS_Subscriber_set_qos** (p. 592), **DDS_Subscriber_get_qos** (p. 593) , **idref_Subscriber_set_qos_with_profile**
- **DDS_Subscriber_set_listener** (p. 597), **DDS_Subscriber_get_listener** (p. 597),
- **DDS_Entity_get_statuscondition** (p. 1365), **DDS_Entity_get_status_changes** (p. 1365)
- **DDS_Subscriber_create_datareader** (p. 581), **DDS_Subscriber_create_datareader_with_profile** (p. 583), **DDS_Subscriber_delete_contained_entities** (p. 585), **DDS_Subscriber_delete_datareader** (p. 585),
- **DDS_Subscriber_set_default_datareader_qos** (p. 580), **DDS_Subscriber_set_default_datareader_qos_with_profile** (p. 580), **DDS_Subscriber_get_default_datareader_qos** (p. 579), **DDS_Subscriber_set_default_library** (p. 595), **DDS_Subscriber_set_default_profile** (p. 594)

All operations except for **DDS_Subscriber_set_qos** (p. 592), **DDS_Subscriber_set_qos_with_profile** (p. 592), **DDS_Subscriber_get_qos** (p. 593), **DDS_Subscriber_set_listener** (p. 597), **DDS_Subscriber_get_listener** (p. 597), **DDS_Entity_enable** (p. 1363) and **DDS_Subscriber_create_datareader** (p. 581) may fail with **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

Operations Allowed in Listener Callbacks (p. 1818)

5.20.3.2 DDS_SubscriberListener_DataOnReadersCallback

```
typedef void(* DDS_SubscriberListener_DataOnReadersCallback) (void *listener_data, DDS_Subscriber *sub)
```

Prototype of a **DDS_SubscriberListener** (p. 2009) `data_on_readers` function.

5.20.4 Function Documentation

5.20.4.1 DDS_SubscriberQos_equals()

```
DDS_Boolean DDS_SubscriberQos_equals (
    const struct DDS_SubscriberQos * self,
    const struct DDS_SubscriberQos * other )
```

Compares two **DDS_SubscriberQos** (p. 2010) for equality.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) This SubscriberQos.
<i>other</i>	<< <i>in</i> >> (p. 1013) The other SubscriberQos to be compared with this SubscriberQos.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two Qos objects are equal or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.20.4.2 DDS_SubscriberQos_print()

```
DDS_ReturnCode_t DDS_SubscriberQos_print (
    const struct DDS_SubscriberQos * self )
```

Prints this **DDS_SubscriberQos** (p. 2010) to stdout.

Only the differences between this **DDS_SubscriberQos** (p. 2010) and the documented default are printed. If you wish to print everything regardless, see **DDS_SubscriberQos_to_string_w_params** (p. 576). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1926).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_RETCODE_OK (p. 1222) if no error was encountered.

5.20.4.3 DDS_SubscriberQos_to_string()

```
DDS_ReturnCode_t DDS_SubscriberQos_to_string (
    const struct DDS_SubscriberQos * self,
    char * string,
    DDS_UnsignedLong * string_size )
```

Obtains a string representation of this **DDS_SubscriberQos** (p. 2010).

Only the differences between this **DDS_SubscriberQos** (p. 2010) and the documented default are printed to the string. If you wish to print everything regardless, see **DDS_SubscriberQos_to_string_w_params** (p. 576). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1926).

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. 2010) is written to the buffer.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>string</i>	<< <i>out</i> >> (p. 1013) The char buffer that will be used to store the string representation of this DDS_SubscriberQos (p. 2010). If NULL, this function will return the required length of this buffer through the <i>string_size</i> parameter.
<i>string_size</i>	<< <i>inout</i> >> (p. 1013) 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. 1222) if no error was encountered.

See also

DDS_SubscriberQos_to_string_w_params (p. 576)

5.20.4.4 DDS_SubscriberQos_to_string_w_params()

```
DDS_ReturnCode_t DDS_SubscriberQos_to_string_w_params (
    const struct DDS_SubscriberQos * self,
    char * string,
    DDS_UnsignedLong * string_size,
    const struct DDS_SubscriberQos * base,
    const struct DDS_QosPrintFormat * format )
```

Obtains a string representation of this **DDS_SubscriberQos** (p. 2010).

Only the differences between this **DDS_SubscriberQos** (p. 2010) and the **DDS_SubscriberQos** (p. 2010) supplied as the base are printed to the string.

It is possible to supply the sentinel value **DDS_SUBSCRIBER_QOS_PRINT_ALL** (p. 164) as the base to print everything within the QoS.

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. 2010) is written to the buffer.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>string</i>	<< <i>out</i> >> (p. 1013) The char buffer that will be used to store the string representation of this DDS_SubscriberQos (p. 2010). If NULL, this function will return the required length of this buffer through the <i>string_size</i> parameter.
<i>string_size</i>	<< <i>inout</i> >> (p. 1013) 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.
<i>base</i>	<< <i>in</i> >> (p. 1013) The DDS_SubscriberQos (p. 2010) to be used as the base QoS profile. Only the differences with respect to this base profile will be included in the output string.
<i>format</i>	<< <i>in</i> >> (p. 1013) The DDS_QosPrintFormat (p. 1926) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 1222) if no error was encountered.

5.20.4.5 DDS_SubscriberQos_initialize()

```
DDS_ReturnCode_t DDS_SubscriberQos_initialize (
    struct DDS_SubscriberQos * self )
```

Initializer for new QoS instances.

New **DDS_SubscriberQos** (p. 2010) instances on heap should be initialized with this function before they are passed to any function. This step ensures that those contained QoS policies that use dynamic memory are properly initialized. This function does not allocate memory.

Calling this function is not a substitute for calling **DDS_Subscriber_get_qos** (p. 593) or **DDS_DomainParticipant_get_default_subscriber_qos** (p. 94); one of those should be called subsequently to setting the QoS of any new or existing entity. **DDS_SubscriberQos_finalize** (p. 577) should be called to free the contained QoS policies that use dynamic memory:

```
DDS_SubscriberQos *myQos = malloc(sizeof(struct DDS_SubscriberQos));

DDS_SubscriberQos_initialize(myQos);

DDS_DomainParticipantFactory_get_default_subscriber_qos(myFactory, myQos);

DDS_Subscriber_set_qos(mySubscriber, myQos);

DDS_SubscriberQos_finalize(myQos);

free(myQos);
```

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_SubscriberQos_INITIALIZER (p. 571)

DDS_DomainParticipant_get_default_subscriber_qos (p. 94)

DDS_SubscriberQos_finalize (p. 577)

5.20.4.6 DDS_SubscriberQos_finalize()

```
DDS_ReturnCode_t DDS_SubscriberQos_finalize (
    struct DDS_SubscriberQos * self )
```

Free any dynamic memory allocated by the policies in this **DDS_SubscriberQos** (p. 2010).

Some QoS policies may use dynamic memory (regardless of whether the QoS itself is in dynamic memory). This function frees that memory but otherwise leaves this QoS unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

This function does not leave this object in an invalid state. It is permissible to clear a QoS and then subsequently allocate new dynamic memory in one or more of its QoS policies.

Note that if this QoS instance is stored in heap memory, calling this function will *not* call `free()` on it; the user is responsible for explicitly freeing any heap-based QoS instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.20.4.7 DDS_SubscriberQos_copy()

```
DDS_ReturnCode_t DDS_SubscriberQos_copy (
    struct DDS_SubscriberQos * self,
    const struct DDS_SubscriberQos * source )
```

Copy the contents of the given QoS into this QoS.

DDS_SubscriberQos (p. 2010) instances can use dynamic memory because of the sequences contained in some QoS policies. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013) The DDS_PublisherQos (p. 1918) to copy from.

Returns

One of the **Standard Return Codes** (p. 1221)

5.20.4.8 DDS_Subscriber_as_entity()

```
DDS_Entity * DDS_Subscriber_as_entity (
    DDS_Subscriber * subscriber )
```

Access a **DDS_Subscriber** (p. 572)'s supertype instance.

Parameters

<i>subscriber</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

The **DDS_Entity** (p. 1360) that is supertype instance of the subscriber.

5.20.4.9 DDS_Subscriber_get_default_datareader_qos()

```
DDS_ReturnCode_t DDS_Subscriber_get_default_datareader_qos (
    DDS_Subscriber * self,
    struct DDS_DataReaderQos * qos )
```

Copies the default **DDS_DataReaderQos** (p. 1611) values into the provided **DDS_DataReaderQos** (p. 1611) instance.

The retrieved *qos* will match the set of values specified on the last successful call to **DDS_Subscriber_set_default_datareader_qos** (p. 580), or **DDS_Subscriber_set_default_datareader_qos_with_profile** (p. 580), or else, if the call was never made, the default values from its owning **DDS_DomainParticipant** (p. 77).

This function 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 **DDS_Subscriber_set_default_datareader_qos** (p. 580)

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>inout</i> >> (p. 1013) DDS_DataReaderQos (p. 1611) to be filled-up. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DATAREADER_QOS_DEFAULT (p. 600)

DDS_Subscriber_create_datareader (p. 581)

5.20.4.10 DDS_Subscriber_set_default_datareader_qos()

```
DDS_ReturnCode_t DDS_Subscriber_set_default_datareader_qos (
    DDS_Subscriber * self,
    const struct DDS_DataReaderQos * qos )
```

Sets the default **DDS_DataReaderQos** (p. 1611) values for this subscriber.

This call causes the default values inherited from the owning **DDS_DomainParticipant** (p. 77) to be overridden.

This default value will be used for newly created **DDS_DataReader** (p. 615) if **DDS_DATAREADER_QOS_DEFAULT** (p. 600) is specified as the `qos` parameter when **DDS_Subscriber_create_datareader** (p. 581) 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. 1222)

MT Safety:

UNSAFE. It is not safe to set the default QoS value from a subscriber while another thread may be simultaneously calling **DDS_Subscriber_set_default_datareader_qos** (p. 580), **DDS_Subscriber_get_default_datareader_↵ qos** (p. 579) or calling **DDS_Subscriber_create_datareader** (p. 581) with **DDS_DATAREADER_QOS_DEFAULT** (p. 600) as the `qos` parameter.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>in</i> >> (p. 1013) The default DDS_DataReaderQos (p. 1611) to be set to. The special value DDS_DATAREADER_QOS_DEFAULT (p. 600) may be passed as <code>qos</code> to indicate that the default QoS should be reset back to the initial values the factory would use if DDS_Subscriber_set_default_datareader_qos (p. 580) had never been called. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), or or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222)

5.20.4.11 DDS_Subscriber_set_default_datareader_qos_with_profile()

```
DDS_ReturnCode_t DDS_Subscriber_set_default_datareader_qos_with_profile (
    DDS_Subscriber * self,
    const char * library_name,
    const char * profile_name )
```

<<**extension**>> (p. 1012) Set the default **DDS_DataReaderQos** (p. 1611) values for this subscriber based on the input XML QoS profile.

This default value will be used for newly created **DDS_DataReader** (p. 615) if **DDS_DATAREADER_QOS_DEFAULT** (p. 600) is specified as the `qos` parameter when **DDS_Subscriber_create_datareader** (p. 581) is called.

Precondition

The **DDS_DataReaderQos** (p. 1611) 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. 1222)

MT Safety:

UNSAFE. It is not safe to set the default QoS value from a **DDS_Subscriber** (p. 572) while another thread may be simultaneously calling **DDS_Subscriber_set_default_datareader_qos** (p. 580), **DDS_Subscriber_get_default_datareader_qos** (p. 579) or calling **DDS_Subscriber_create_datareader** (p. 581) with **DDS_DATAREADER_QOS_DEFAULT** (p. 600) as the `qos` parameter.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) Library name containing the XML QoS profile. If <code>library_name</code> is null RTI Connexx will use the default library (see DDS_Subscriber_set_default_library (p. 595)).
<i>profile_name</i>	<< in >> (p. 1013) XML QoS Profile name. If <code>profile_name</code> is null RTI Connexx will use the default profile (see DDS_Subscriber_set_default_profile (p. 594)).

If the input profile cannot be found the function fails with **DDS_RETCODE_ERROR** (p. 1222).

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222)

See also

DDS_DATAREADER_QOS_DEFAULT (p. 600)

DDS_Subscriber_create_datareader_with_profile (p. 583)

5.20.4.12 DDS_Subscriber_create_datareader()

```
DDS_DataReader * DDS_Subscriber_create_datareader (
    DDS_Subscriber * self,
    DDS_TopicDescription * topic,
    const struct DDS_DataReaderQos * qos,
    const struct DDS_DataReaderListener * listener,
    DDS_StatusMask mask )
```

Creates a **DDS_DataReader** (p. 615) that will be attached and belong to the **DDS_Subscriber** (p. 572).

For each application-defined type **Foo** (p. 2109), there is an implied, auto-generated class **FooDataReader** (p. 2112) (an incarnation of **FooDataReader** (p. 2112)) that extends **DDS_DataReader** (p. 615) and contains the operations to read data of type **Foo** (p. 2109).

Note that a common application pattern to construct the QoS for the **DDS_DataReader** (p. 615) is to:

- Retrieve the QoS policies on the associated **DDS_Topic** (p. 175) by means of the **DDS_Topic_get_qos** (p. 199) operation.
- Retrieve the default **DDS_DataReader** (p. 615) qos by means of the **DDS_Subscriber_get_default_datareader_qos** (p. 579) operation.
- Combine those two QoS policies (for example, using **DDS_Subscriber_copy_from_topic_qos** (p. 591)) and selectively modify policies as desired
- Use the resulting QoS policies to construct the **DDS_DataReader** (p. 615).

When a **DDS_DataReader** (p. 615) is created, only those transports already registered are available to the **DDS_DataReader** (p. 615). See **Built-in Transport Plugins** (p. 736) for details on when a builtin transport is registered.

MT Safety:

UNSAFE. If **DDS_DATAREADER_QOS_DEFAULT** (p. 600) is used for the `qos` parameter, it is not safe to create the datareader while another thread may be simultaneously calling **DDS_Subscriber_set_default_datareader_qos** (p. 580).

Precondition

If subscriber is enabled, the topic must be enabled. If it is not, this operation will fail and no **DDS_DataReader** (p. 615) will be created.

The given **DDS_TopicDescription** (p. 175) must have been created from the same participant as this subscriber. If it was created from a different participant, this function will return NULL.

If `qos` is **DDS_DATAREADER_QOS_USE_TOPIC_QOS** (p. 601), `topic` cannot be **DDS_MultiTopic** (p. 185), or else this function will return NULL.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>topic</i>	<< <i>in</i> >> (p. 1013) The DDS_TopicDescription (p. 175) that the DDS_DataReader (p. 615) will be associated with. Cannot be NULL.

Parameters

<i>qos</i>	<< <i>in</i> >> (p. 1013) The qos of the DDS_DataReader (p. 615). The special value DDS_DATAREADER_QOS_DEFAULT (p. 600) can be used to indicate that the DDS_DataReader (p. 615) should be created with the default DDS_DataReaderQos (p. 1611) set in the DDS_Subscriber (p. 572). If DDS_TopicDescription (p. 175) is of type DDS_Topic (p. 175) or DDS_ContentFilteredTopic (p. 177), the special value DDS_DATAREADER_QOS_USE_TOPIC_QOS (p. 601) can be used to indicate that the DDS_DataReader (p. 615) should be created with the combination of the default DDS_DataReaderQos (p. 1611) set on the DDS_Subscriber (p. 572) and the DDS_TopicQos (p. 2042) (in the case of a DDS_ContentFilteredTopic (p. 177), the DDS_TopicQos (p. 2042) of the related DDS_Topic (p. 175)). if DDS_DATAREADER_QOS_USE_TOPIC_QOS (p. 601) is used, <i>topic</i> cannot be a DDS_MultiTopic (p. 185). Cannot be NULL.
<i>listener</i>	<< <i>in</i> >> (p. 1013) The listener of the DDS_DataReader (p. 615).
<i>mask</i>	<< <i>in</i> >> (p. 1013). Changes of communication status to be invoked on the listener. See DDS_StatusMask (p. 1226).

Returns

A **DDS_DataReader** (p. 615) of a derived class specific to the data-type associated with the **DDS_Topic** (p. 175) or NULL if an error occurred.

See also

FooDataReader (p. 2112)
Specifying QoS on entities (p. 1244) for information on setting QoS before entity creation
DDS_DataReaderQos (p. 1611) for rules on consistency among QoS
DDS_Subscriber_create_datareader_with_profile (p. 583)
DDS_Subscriber_get_default_datareader_qos (p. 579)
DDS_Topic_set_qos (p. 197)
DDS_Subscriber_copy_from_topic_qos (p. 591)
DDS_DataReader_set_listener (p. 690)

Examples

HelloWorld_subscriber.c.

5.20.4.13 DDS_Subscriber_create_datareader_with_profile()

```

DDS_DataReader * DDS_Subscriber_create_datareader_with_profile (
    DDS_Subscriber * self,
    DDS_TopicDescription * topic,
    const char * library_name,
    const char * profile_name,
    const struct DDS_DataReaderListener * listener,
    DDS_StatusMask mask )

```

<<*extension*>> (p. 1012) Creates a **DDS_DataReader** (p. 615) object using the **DDS_DataReaderQos** (p. 1611) associated with the input XML QoS profile.

The **DDS_DataReader** (p. 615) will be attached and belong to the **DDS_Subscriber** (p. 572).

For each application-defined type **Foo** (p. 2109), there is an implied, auto-generated class **FooDataReader** (p. 2112) (an incarnation of **FooDataReader** (p. 2112)) that extends **DDS_DataReader** (p. 615) and contains the operations to read data of type **Foo** (p. 2109).

When a **DDS_DataReader** (p. 615) is created, only those transports already registered are available to the **DDS_DataReader** (p. 615). See **Built-in Transport Plugins** (p. 736) 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 **DDS_DataReader** (p. 615) will be created.

The given **DDS_TopicDescription** (p. 175) must have been created from the same participant as this subscriber. If it was created from a different participant, this function will return NULL.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>topic</i>	<< <i>in</i> >> (p. 1013) The DDS_TopicDescription (p. 175) that the DDS_DataReader (p. 615) will be associated with. Cannot be NULL.
<i>library_name</i>	<< <i>in</i> >> (p. 1013) Library name containing the XML QoS profile. If library_name is null RTI Connex will use the default library (see DDS_Subscriber_set_default_library (p. 595)).
<i>profile_name</i>	<< <i>in</i> >> (p. 1013) XML QoS Profile name. If profile_name is null RTI Connex will use the default profile (see DDS_Subscriber_set_default_profile (p. 594)).
<i>listener</i>	<< <i>in</i> >> (p. 1013) The listener of the DDS_DataReader (p. 615).
<i>mask</i>	<< <i>in</i> >> (p. 1013). Changes of communication status to be invoked on the listener. See DDS_StatusMask (p. 1226).

Returns

A **DDS_DataReader** (p. 615) of a derived class specific to the data-type associated with the **DDS_Topic** (p. 175) or NULL if an error occurred.

See also

FooDataReader (p. 2112)

Specifying QoS on entities (p. 1244) for information on setting QoS before entity creation

DDS_DataReaderQos (p. 1611) for rules on consistency among QoS

DDS_DATAREADER_QOS_DEFAULT (p. 600)

DDS_DATAREADER_QOS_USE_TOPIC_QOS (p. 601)

DDS_Subscriber_create_datareader (p. 581)

DDS_Subscriber_get_default_datareader_qos (p. 579)

DDS_Topic_set_qos (p. 197)

DDS_Subscriber_copy_from_topic_qos (p. 591)

DDS_DataReader_set_listener (p. 690)

5.20.4.14 DDS_Subscriber_delete_datareader()

```
DDS_ReturnCode_t DDS_Subscriber_delete_datareader (
    DDS_Subscriber * self,
    DDS_DataReader * a_datareader )
```

Deletes a **DDS_DataReader** (p. 615) that belongs to the **DDS_Subscriber** (p. 572).

Precondition

If the **DDS_DataReader** (p. 615) does not belong to the **DDS_Subscriber** (p. 572), or if there are any existing **DDS_ReadCondition** (p. 694) or **DDS_QueryCondition** (p. 699) objects that are attached to the **DDS_DataReader** (p. 615), 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. 1222).

Postcondition

Listener installed on the **DDS_DataReader** (p. 615) will not be called after this function 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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>a_datareader</i>	<< <i>in</i> >> (p. 1013) The DDS_DataReader (p. 615) to be deleted.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

5.20.4.15 DDS_Subscriber_delete_contained_entities()

```
DDS_ReturnCode_t DDS_Subscriber_delete_contained_entities (
    DDS_Subscriber * self )
```

Deletes all the entities that were created by means of the "create" operation on the **DDS_Subscriber** (p. 572).

Deletes all contained **DDS_DataReader** (p. 615) objects. This pattern is applied recursively. In this manner, the operation **DDS_Subscriber_delete_contained_entities** (p. 585) on the **DDS_Subscriber** (p. 572) will end up deleting all the

entities recursively contained in the **DDS_Subscriber** (p. 572), including the **DDS_QueryCondition** (p. 699), **DDS_ReadCondition** (p. 694), and **DDS_TopicQuery** (p. 707) objects belonging to the contained **DDS_DataReader** (p. 615).

The operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) if any of the contained entities is in a state where it cannot be deleted. This will occur, for example, if a contained **DDS_DataReader** (p. 615) cannot be deleted because the application has called a **FooDataReader_read** (p. 625) or **FooDataReader_take** (p. 626) operation and has not called the corresponding **FooDataReader_return_loan** (p. 646) operation to return the loaned samples.

Once **DDS_Subscriber_delete_contained_entities** (p. 585) completes successfully, the application may delete the **DDS_Subscriber** (p. 572).

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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222)

5.20.4.16 DDS_Subscriber_lookup_datareader()

```
DDS_DataReader * DDS_Subscriber_lookup_datareader (
    DDS_Subscriber * self,
    const char * topic_name )
```

Retrieves an existing **DDS_DataReader** (p. 615).

Use this operation on the built-in **DDS_Subscriber** (p. 572) (**Built-in Topics** (p. 165)) to access the built-in **DDS_DataReader** (p. 615) entities for the built-in topics.

The built-in **DDS_DataReader** (p. 615) is created when this operation is called on a built-in topic for the first time. The built-in **DDS_DataReader** (p. 615) is deleted automatically when the **DDS_DomainParticipant** (p. 77) is deleted.

To ensure that builtin **DDS_DataReader** (p. 615) entities receive all the discovery traffic, it is suggested that you lookup the builtin **DDS_DataReader** (p. 615) before the **DDS_DomainParticipant** (p. 77) is enabled. Looking up builtin **DDS_DataReader** (p. 615) may implicitly register builtin transports due to creation of **DDS_DataReader** (p. 615) (see **Built-in Transport Plugins** (p. 736) 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 **DDS_DomainParticipant** (p. 77).
- (optional) Modify builtin transport properties
- Call **DDS_DomainParticipant_get_builtin_subscriber()** (p. 129).
- Call **DDS_Subscriber_lookup_datareader()** (p. 586).
- Call **enable()** on the DomainParticipant.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>topic_name</i>	<< <i>in</i> >> (p. 1013) Name of the DDS_TopicDescription (p. 175) that the retrieved DDS_DataReader (p. 615) is attached to. Cannot be NULL.

Returns

A **DDS_DataReader** (p. 615) that belongs to the **DDS_Subscriber** (p. 572) attached to the **DDS_TopicDescription** (p. 175) with *topic_name*. If no such **DDS_DataReader** (p. 615) exists, this operation returns NULL.

The returned **DDS_DataReader** (p. 615) may be enabled or disabled.

If more than one **DDS_DataReader** (p. 615) is attached to the **DDS_Subscriber** (p. 572), this operation may return any one of them.

MT Safety:

UNSAFE. It is not safe to lookup a **DDS_DataReader** (p. 615) in one thread while another thread is simultaneously creating or destroying that **DDS_DataReader** (p. 615).

5.20.4.17 DDS_Subscriber_begin_access()

```
DDS_ReturnCode_t DDS_Subscriber_begin_access (
    DDS_Subscriber * self )
```

Indicates that the application is about to access the data samples in any of the **DDS_DataReader** (p. 615) objects attached to the **DDS_Subscriber** (p. 572).

If the **DDS_PresentationQosPolicy::access_scope** (p. 1893) of the **DDS_Subscriber** (p. 572) is **DDS_GROUP_PRESENTATION_QOS** (p. 1305) or **DDS_HIGHEST_OFFERED_PRESENTATION_QOS** (p. 1305) and **DDS_PresentationQosPolicy::ordered_access** (p. 1893) is **DDS_BOOLEAN_TRUE** (p. 1201), the application is required to use this operation to access the samples in order across DataWriters of the same group (**DDS_Publisher** (p. 443) with **DDS_PresentationQosPolicy::access_scope** (p. 1893) set to **DDS_GROUP_PRESENTATION_QOS** (p. 1305)).

In the above case, this operation must be called prior to calling any of the sample-accessing operations, **DDS_Subscriber_get_datareaders** (p. 588) on the **DDS_Subscriber** (p. 572), and **FooDataReader_read** (p. 625), **FooDataReader_take** (p. 626), **FooDataReader_read_w_condition** (p. 630), and **FooDataReader_take_w_condition** (p. 632) on any **DDS_DataReader** (p. 615).

Once the application has finished accessing the data samples, it must call **DDS_Subscriber_end_access** (p. 588)

The application is not required to call **DDS_Subscriber_begin_access** (p. 587) / **DDS_Subscriber_end_access** (p. 588) to access the samples in order if the **PRESENTATION** (p. 1304) policy in the **DDS_Publisher** (p. 443) has **DDS_PresentationQosPolicy::access_scope** (p. 1893) set to something other than **DDS_GROUP_PRESENTATION_QOS** (p. 1305). In this case, calling **DDS_Subscriber_begin_access** (p. 587) / **DDS_Subscriber_end_access** (p. 588) is not considered an error and has no effect.

Calls to **DDS_Subscriber_begin_access** (p. 587) / **DDS_Subscriber_end_access** (p. 588) may be nested and must be balanced.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

Access to data samples (p. 567)

DDS_Subscriber_get_datareaders (p. 588)

PRESENTATION (p. 1304)

5.20.4.18 DDS_Subscriber_end_access()

```
DDS_ReturnCode_t DDS_Subscriber_end_access (
    DDS_Subscriber * self )
```

Indicates that the application has finished accessing the data samples in **DDS_DataReader** (p. 615) objects managed by the **DDS_Subscriber** (p. 572).

This operation must be used to close a corresponding `begin_access()`.

This call must close a previous call to **DDS_Subscriber_begin_access()** (p. 587), otherwise the operation will fail with the error **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

5.20.4.19 DDS_Subscriber_get_datareaders()

```
DDS_ReturnCode_t DDS_Subscriber_get_datareaders (
    DDS_Subscriber * self,
```

```
struct DDS_DataReaderSeq * readers,
    DDS_SampleStateMask sample_states,
    DDS_ViewStateMask view_states,
    DDS_InstanceStateMask instance_states )
```

Allows the application to access the **DDS_DataReader** (p.615) objects that contain samples with the specified `sample_states`, `view_states` and `instance_states`.

If the application is outside a `begin_access()/end_access()` block, or if the **DDS_PresentationQosPolicy::access_scope** (p.1893) of the **DDS_Subscriber** (p.572) is **DDS_INSTANCE_PRESENTATION_QOS** (p.1305) or **DDS_TOPIC_PRESENTATION_QOS** (p.1305), or if the **DDS_PresentationQosPolicy::ordered_access** (p.1893) of the **DDS_Subscriber** (p.572) is **DDS_BOOLEAN_FALSE** (p.1201), the returned collection is a 'set' containing each **DDS_DataReader** (p.615) at most once, in no specified order.

If the application is within a `begin_access()/end_access()` block, and the **PRESENTATION** (p.1304) policy of the **DDS_Subscriber** (p.572) is **DDS_GROUP_PRESENTATION_QOS** (p.1305) or **DDS_HIGHEST_OFFERED_PRESENTATION_QOS** (p.1305), and **DDS_PresentationQosPolicy::ordered_access** (p.1893) in the **DDS_Subscriber** (p.572) is **DDS_BOOLEAN_TRUE** (p.1201), 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 (**DDS_Publisher** (p.443) configured with **DDS_GROUP_PRESENTATION_QOS** (p.1305)), 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.1222).

Parameters

<i>self</i>	<< <i>in</i> >> (p.1013) Cannot be NULL.
<i>readers</i>	<< <i>inout</i> >> (p.1013) a DDS_DataReaderSeq (p.1637) object where the set or list of readers will be returned. Cannot be NULL.
<i>sample_states</i>	<< <i>in</i> >> (p.1013) the returned DataReader must contain samples that have one of these <code>sample_states</code> .
<i>view_states</i>	<< <i>in</i> >> (p.1013) the returned DataReader must contain samples that have one of these <code>view_states</code> .
<i>instance_states</i>	<< <i>in</i> >> (p.1013) the returned DataReader must contain samples that have one of these <code>instance_states</code> .

Returns

One of the **Standard Return Codes** (p.1221) or **DDS_RETCODE_NOT_ENABLED** (p.1222).

See also

Access to data samples (p.567)
DDS_Subscriber_begin_access (p.587)
DDS_Subscriber_end_access (p.588)
PRESENTATION (p.1304)

5.20.4.20 DDS_Subscriber_get_all_datareaders()

```
DDS_ReturnCode_t DDS_Subscriber_get_all_datareaders (
    DDS_Subscriber * self,
    struct DDS_DataReaderSeq * readers )
```

Retrieve all the DataReaders created from this Subscriber.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>readers</i>	<< <i>inout</i> >> (p. 1013) Sequence where the DataReaders will be added

Returns

One of the **Standard Return Codes** (p. 1221)

5.20.4.21 DDS_Subscriber_notify_datareaders()

```
DDS_ReturnCode_t DDS_Subscriber_notify_datareaders (
    DDS_Subscriber * self )
```

Invokes the operation **DDS_DataReaderListener::on_data_available()** (p. 1595) on the **DDS_DataReaderListener** (p. 1593) objects attached to contained **DDS_DataReader** (p. 615) entities with **DDS_DATA_AVAILABLE_STATUS** (p. 1230) that is considered changed as described in **Changes in read communication status** (p. 1225).

This operation is typically invoked from the **DDS_SubscriberListener::on_data_on_readers** (p. 2010) operation in the **DDS_SubscriberListener** (p. 2009). That way the **DDS_SubscriberListener** (p. 2009) can delegate to the **DDS_↵DataReaderListener** (p. 1593) 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. 711), `view_state` of **DDS_ANY_SAMPLE_STATE** (p. 711) and `instance_state` of **DDS_ANY_INSTANCE_STATE** (p. 715).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

5.20.4.22 DDS_Subscriber_get_participant()

```
DDS_DomainParticipant * DDS_Subscriber_get_participant (
    DDS_Subscriber * self )
```

Returns the **DDS_DomainParticipant** (p. 77) to which the **DDS_Subscriber** (p. 572) belongs.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

the **DDS_DomainParticipant** (p. 77) to which the **DDS_Subscriber** (p. 572) belongs.

5.20.4.23 DDS_Subscriber_copy_from_topic_qos()

```
DDS_ReturnCode_t DDS_Subscriber_copy_from_topic_qos (
    DDS_Subscriber * self,
    struct DDS_DataReaderQos * datareader_qos,
    const struct DDS_TopicQos * topic_qos )
```

Copies the policies in the **DDS_TopicQos** (p. 2042) to the corresponding policies in the **DDS_DataReaderQos** (p. 1611).

Copies the policies in the **DDS_TopicQos** (p. 2042) to the corresponding policies in the **DDS_DataReaderQos** (p. 1611) (replacing values in the **DDS_DataReaderQos** (p. 1611), if present).

This is a "convenience" operation most useful in combination with the operations **DDS_Subscriber_get_default_datareader_qos** (p. 579) and **DDS_Topic_get_qos** (p. 199). The operation **DDS_Subscriber_copy_from_topic_qos** (p. 591) can be used to merge the **DDS_DataReader** (p. 615) default QoS policies with the corresponding ones on the **DDS_Topic** (p. 175). The resulting QoS can then be used to create a new **DDS_DataReader** (p. 615), or set its QoS.

This operation does not check the resulting **DDS_DataReaderQos** (p. 1611) for consistency. This is because the 'merged' **DDS_DataReaderQos** (p. 1611) may not be the final one, as the application can still modify some policies prior to applying the policies to the **DDS_DataReader** (p. 615).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>datareader_qos</i>	<< <i>inout</i> >> (p. 1013) DDS_DataReaderQos (p. 1611) to be filled-up. Cannot be NULL.
<i>topic_qos</i>	<< <i>in</i> >> (p. 1013) DDS_TopicQos (p. 2042) to be merged with DDS_DataReaderQos (p. 1611). Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.20.4.24 DDS_Subscriber_set_qos()

```
DDS_ReturnCode_t DDS_Subscriber_set_qos (
    DDS_Subscriber * self,
    const struct DDS_SubscriberQos * qos )
```

Sets the subscriber QoS.

This operation modifies the QoS of the **DDS_Subscriber** (p. 572).

The **DDS_SubscriberQos::group_data** (p. 2011), **DDS_SubscriberQos::partition** (p. 2011) and **DDS_SubscriberQos::entity_factory** (p. 2011) can be changed. The other policies are immutable.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>in</i> >> (p. 1013) DDS_SubscriberQos (p. 2010) to be set to. Policies must be consistent. Immutable policies cannot be changed after DDS_Subscriber (p. 572) is enabled. The special value DDS_SUBSCRIBER_QOS_DEFAULT (p. 162) can be used to indicate that the QoS of the DDS_Subscriber (p. 572) should be changed to match the current default DDS_SubscriberQos (p. 2010) set in the DDS_DomainParticipant (p. 77). Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 1222), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222).

See also

DDS_SubscriberQos (p. 2010) for rules on consistency among QoS

set_qos (abstract) (p. 1361)

Operations Allowed in Listener Callbacks (p. 1818)

5.20.4.25 DDS_Subscriber_set_qos_with_profile()

```
DDS_ReturnCode_t DDS_Subscriber_set_qos_with_profile (
    DDS_Subscriber * self,
```



```
const char * library_name,
const char * profile_name )
```

<<**extension**>> (p. 1012) Change the QoS of this subscriber using the input XML QoS profile.

This operation modifies the QoS of the **DDS_Subscriber** (p. 572).

The **DDS_SubscriberQos::group_data** (p. 2011), **DDS_SubscriberQos::partition** (p. 2011) and **DDS_SubscriberQos::entity_factory** (p. 2011) can be changed. The other policies are immutable.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) Library name containing the XML QoS profile. If library_name is null RTI Connex will use the default library (see DDS_Subscriber_set_default_library (p. 595)).
<i>profile_name</i>	<< in >> (p. 1013) XML QoS Profile name. If profile_name is null RTI Connex will use the default profile (see DDS_Subscriber_set_default_profile (p. 594)).

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 1222), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222).

See also

DDS_SubscriberQos (p. 2010) for rules on consistency among QoS
Operations Allowed in Listener Callbacks (p. 1818)

5.20.4.26 DDS_Subscriber_get_qos()

```
DDS_ReturnCode_t DDS_Subscriber_get_qos (
    DDS_Subscriber * self,
    struct DDS_SubscriberQos * qos )
```

Gets the subscriber QoS.

This function may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< in >> (p. 1013) DDS_SubscriberQos (p. 2010) to be filled in. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

get_qos (abstract) (p. 1362)

5.20.4.27 DDS_Subscriber_set_default_profile()

```
DDS_ReturnCode_t DDS_Subscriber_set_default_profile (
    DDS_Subscriber * self,
    const char * library_name,
    const char * profile_name )
```

<<**extension**>> (p. 1012) Sets the default XML profile for a **DDS_Subscriber** (p. 572).

This function 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 **DDS_Subscriber** (p. 572) function 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 **DDS_Subscriber** (p. 572) inherits the default from the **DDS_DomainParticipant** (p. 77) (see **DDS_DomainParticipant_set_default_profile** (p. 102)).

This function does not set the default QoS for **DDS_DataReader** (p. 615) objects created by this **DDS_Subscriber** (p. 572); for this functionality, use **DDS_Subscriber_set_default_datareader_qos_with_profile** (p. 580) (you may pass in NULL after having called `set_default_profile()`).

This function does not set the default QoS for newly created Subscribers; for this functionality, use **DDS_DomainParticipant_set_default_subscriber_qos_with_profile** (p. 95).

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) The library name containing the profile.
<i>profile_name</i>	<< in >> (p. 1013) The profile name. If <code>profile_name</code> is null any previous default is unset.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_Subscriber_get_default_profile (p. 594)

DDS_Subscriber_get_default_profile_library (p. 595)

5.20.4.28 DDS_Subscriber_get_default_profile()

```
const char * DDS_Subscriber_get_default_profile (
    DDS_Subscriber * self )
```

<<**extension**>> (p. 1012) Gets the default XML profile associated with a **DDS_Subscriber** (p. 572).

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.

Returns

The default profile or null if the default profile was not set.

See also

DDS_Subscriber_set_default_profile (p. 594)

5.20.4.29 DDS_Subscriber_get_default_profile_library()

```
const char * DDS_Subscriber_get_default_profile_library (
    DDS_Subscriber * self )
```

<<**extension**>> (p. 1012) Gets the library where the default XML QoS profile is contained for a **DDS_Subscriber** (p. 572).

The default profile library is automatically set when **DDS_Subscriber_set_default_profile** (p. 594) is called.

This library can be different than the **DDS_Subscriber** (p. 572) default library (see **DDS_Subscriber_get_default_library** (p. 596)).

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.

Returns

The default profile library or null if the default profile was not set.

See also

DDS_Subscriber_set_default_profile (p. 594)

5.20.4.30 DDS_Subscriber_set_default_library()

```
DDS_ReturnCode_t DDS_Subscriber_set_default_library (
    DDS_Subscriber * self,
    const char * library_name )
```

<<**extension**>> (p. 1012) Sets the default XML library for a **DDS_Subscriber** (p. 572).

This function 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 **DDS_Subscriber** (p. 572) inherits the default from the **DDS_DomainParticipant** (p. 77) (see **DDS_DomainParticipant_set_default_library** (p. 101)).

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) Library name. If library_name is null any previous default is unset.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_Subscriber_get_default_library (p. 596)

5.20.4.31 DDS_Subscriber_get_default_library()

```
const char * DDS_Subscriber_get_default_library (
    DDS_Subscriber * self )
```

<<**extension**>> (p. 1012) Gets the default XML library associated with a **DDS_Subscriber** (p. 572).

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.

Returns

The default library or null if the default library was not set.

See also

DDS_Subscriber_set_default_library (p. 595)

5.20.4.32 DDS_Subscriber_set_listener()

```
DDS_ReturnCode_t DDS_Subscriber_set_listener (
    DDS_Subscriber * self,
    const struct DDS_SubscriberListener * l,
    DDS_StatusMask mask )
```

Sets the subscriber listener.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>l</i>	<< <i>in</i> >> (p. 1013) DDS_SubscriberListener (p. 2009) to set to.
<i>mask</i>	<< <i>in</i> >> (p. 1013) DDS_StatusMask (p. 1226) associated with the DDS_SubscriberListener (p. 2009). The callback function on the listener cannot be NULL if the corresponding status is turned on in the <i>mask</i> .

Returns

One of the **Standard Return Codes** (p. 1221)

See also

set_listener (abstract) (p. 1362)

5.20.4.33 DDS_Subscriber_get_listener()

```
struct DDS_SubscriberListener DDS_Subscriber_get_listener (
    DDS_Subscriber * self )
```

Get the subscriber listener.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_SubscriberListener (p. 2009) of the **DDS_Subscriber** (p. 572).

See also

DDS_Subscriber_get_listenerX (p. 598)

get_listener (abstract) (p. 1363)

5.20.4.34 DDS_Subscriber_get_listenerX()

```
DDS_ReturnCode_t DDS_Subscriber_get_listenerX (
    DDS_Subscriber * self,
    struct DDS_SubscriberListener * listener )
```

<<**extension**>> (p. 1012) Get the subscriber listener.

An alternative form of `get_listener` that fills in an existing listener structure rather than returning one on the stack.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>listener</i>	<< inout >> (p. 1013) DDS_SubscriberListener (p. 2009) structure to be filled up. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_Subscriber_get_listener (p. 597)

get_listener (abstract) (p. 1363)

5.20.4.35 DDS_Subscriber_lookup_datareader_by_name()

```
DDS_DataReader * DDS_Subscriber_lookup_datareader_by_name (
    DDS_Subscriber * self,
    const char * datareader_name )
```

<<**extension**>> (p. 1012) Retrieves a **DDS_DataReader** (p. 615) contained within the **DDS_Subscriber** (p. 572) the **DDS_DataReader** (p. 615) entity name.

Every **DDS_DataReader** (p.615) in the system has an entity name which is configured and stored in the `<<extension>>` (p. 1012) EntityName policy, **ENTITY_NAME** (p. 1290).

This operation retrieves the **DDS_DataReader** (p.615) within the **DDS_Subscriber** (p.572) whose name matches the one specified. If there are several **DDS_DataReader** (p.615) with the same name within the **DDS_Subscriber** (p.572), the operation returns the first matching occurrence.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>datareader_name</i>	<< <i>in</i> >> (p. 1013) Entity name of the DDS_DataReader (p. 615).

Returns

The first **DDS_DataReader** (p. 615) found with the specified name or NULL if it is not found.

See also

DDS_DomainParticipant_lookup_datareader_by_name (p. 159)

5.20.5 Variable Documentation

5.20.5.1 DDS_DATAREADER_QOS_PRINT_ALL

```
const struct DDS_DataReaderQos* DDS_DATAREADER_QOS_PRINT_ALL [extern]
```

Special value which can be supplied to **DDS_DataReaderQos_to_string_w_params** (p. 668) indicating that all of the QoS should be printed.

The **DDS_DataReaderQos_to_string_w_params** (p. 668) API accepts a base QoS profile as one of its arguments. The resultant string only contains the differences with respect to the supplied base QoS profile. Supplying the **DDS_DATAREADER_QOS_PRINT_ALL** (p. 600) sentinel value as the base QoS will result in all of the QoS being printed.

Note that there are some QoS policies and fields which are not intended for public use. Even when **DDS_DATAREADER_QOS_PRINT_ALL** (p. 600) is supplied as the base, these will not be printed unless they differ from the documented default. If you want to see their values, you must use the `print_private` field within the **DDS_QosPrintFormat** (p. 1926) structure.

This value should only be used as the base parameter to the **DDS_DataReaderQos_to_string_w_params** (p. 668) API.

5.20.5.2 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 **DDS_Subscriber_create_datareader** (p. 581), this special value is used to indicate that the **DDS_DataReader** (p. 615) should be created with the default **DDS_DataReader** (p. 615) QoS by means of the operation `get_default_datareader_qos` and using the resulting QoS to create the **DDS_DataReader** (p. 615).

When used in **DDS_Subscriber_set_default_datareader_qos** (p. 580), 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 **DDS_Subscriber_set_default_datareader_qos** (p. 580) operation had never been called.

When used in **DDS_DataReader_set_qos** (p. 687), this special value is used to indicate that the QoS of the **DDS_DataReader** (p. 615) should be changed to match the current default QoS set in the **DDS_Subscriber** (p. 572) that the **DDS_DataReader** (p. 615) 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 **DDS_DomainParticipant_get_default_datareader_qos** (p. 96).

See also

DDS_Subscriber_create_datareader (p. 581)

DDS_Subscriber_set_default_datareader_qos (p. 580)

DDS_DataReader_set_qos (p. 687)

Examples

HelloWorld_subscriber.c.

5.20.5.3 DDS_DATAREADER_QOS_USE_TOPIC_QOS

```
const struct DDS_DataReaderQos DDS_DATAREADER_QOS_USE_TOPIC_QOS [extern]
```

Special value for creating **DDS_DataReader** (p. 615) with a combination of the default **DDS_DataReaderQos** (p. 1611) and the **DDS_TopicQos** (p. 2042).

The use of this value is equivalent to the application obtaining the default **DDS_DataReaderQos** (p. 1611) and the **DDS_TopicQos** (p. 2042) (by means of the operation **DDS_Topic_get_qos** (p. 199)) and then combining these two QoS using the operation **DDS_Subscriber_copy_from_topic_qos** (p. 591) whereby any policy that is set on the **DDS_↔TopicQos** (p. 2042) "overrides" the corresponding policy on the default QoS. The resulting QoS is then applied to the creation of the **DDS_DataReader** (p. 615).

This value should only be used in **DDS_Subscriber_create_datareader** (p. 581).

See also

DDS_Subscriber_create_datareader (p. 581)

DDS_Subscriber_get_default_datareader_qos (p. 579)

DDS_Topic_get_qos (p. 199)

DDS_Subscriber_copy_from_topic_qos (p. 591)

5.21 DataReaders

DDS_DataReader (p. 615) entity and associated elements

Modules

- **Read Conditions**
DDS_ReadCondition (p. 694) and associated elements
- **Query Conditions**
DDS_QueryCondition (p. 699) and associated elements
- **Topic Queries**
DDS_TopicQuery (p. 707) and associated elements.

Data Structures

- struct **FooDataReader**
 <<*interface*>> (p. 1013) <<*generic*>> (p. 1013) User data type-specific data reader.
- struct **DDS_RequestedDeadlineMissedStatus**
 DDS_REQUESTED_DEADLINE_MISSED_STATUS (p. 1229)
- struct **DDS_LivelinessChangedStatus**
 DDS_LIVELINESS_CHANGED_STATUS (p. 1231)
- struct **DDS_RequestedIncompatibleQosStatus**
 DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS (p. 1229)
- struct **DDS_SampleLostStatus**
 DDS_SAMPLE_LOST_STATUS (p. 1230)
- struct **DDS_SampleRejectedStatus**
 DDS_SAMPLE_REJECTED_STATUS (p. 1230)
- struct **DDS_SubscriptionMatchedStatus**
 DDS_SUBSCRIPTION_MATCHED_STATUS (p. 1232)
- struct **DDS_DataReaderCacheStatus**
 <<*extension*>> (p. 1012) The status of the reader's cache.
- struct **DDS_DataReaderProtocolStatus**
 <<*extension*>> (p. 1012) 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 *DDS_DataReader* (p. 615) entity.
- struct **DDS_DataReaderSeq**
 Declares IDL sequence < *DDS_DataReader* (p. 615) > .
- struct **DDS_DataReaderListener**
 <<*interface*>> (p. 1013) *DDS_Listener* (p. 1816) for reader status.

Macros

- #define **DDS_RequestedDeadlineMissedStatus_INITIALIZER**
 Initializer for new status instances.
- #define **DDS_LivelinessChangedStatus_INITIALIZER**
 Initializer for new status instances.
- #define **DDS_RequestedIncompatibleQosStatus_INITIALIZER**
 Initializer for new status instances.
- #define **DDS_SampleLostStatus_INITIALIZER**

- *Initializer for new status instances.*
- **#define DDS_SampleRejectedStatus_INITIALIZER**
Initializer for new status instances.
- **#define DDS_SubscriptionMatchedStatus_INITIALIZER**
Initializer for new status instances.
- **#define DDS_DataReaderCacheStatus_INITIALIZER**
Initializer for new status instances.
- **#define DDS_DataReaderProtocolStatus_INITIALIZER**
Initializer for new status instances.
- **#define DDS_DataReaderQos_INITIALIZER**
Initializer for new QoS instances.
- **#define DDS_DataReaderListener_INITIALIZER**
*Initializer for new **DDS_DataReaderListener** (p. 1593).*

Typedefs

- **typedef struct DDS_DataReaderImpl DDS_DataReader**
*<<interface>> (p. 1013) 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 **DDS_Subscriber** (p. 572).*
- **typedef void(* DDS_DataReaderListener_RequestedDeadlineMissedCallback)** (void *listener_data, **DDS_DataReader** *reader, const struct **DDS_RequestedDeadlineMissedStatus** *status)
*Prototype of a **DDS_DataReaderListener** (p. 1593) requested_deadline_missed function.*
- **typedef void(* DDS_DataReaderListener_LivelinessChangedCallback)** (void *listener_data, **DDS_DataReader** *reader, const struct **DDS_LivelinessChangedStatus** *status)
*Prototype of a **DDS_DataReaderListener** (p. 1593) liveliness_changed function.*
- **typedef void(* DDS_DataReaderListener_RequestedIncompatibleQosCallback)** (void *listener_data, **DDS_DataReader** *reader, const struct **DDS_RequestedIncompatibleQosStatus** *status)
*Prototype of a **DDS_DataReaderListener** (p. 1593) requested_incompatible_QoS function.*
- **typedef void(* DDS_DataReaderListener_SampleRejectedCallback)** (void *listener_data, **DDS_DataReader** *reader, const struct **DDS_SampleRejectedStatus** *status)
*Prototype of a **DDS_DataReaderListener** (p. 1593) sample_rejected function.*
- **typedef void(* DDS_DataReaderListener_DataAvailableCallback)** (void *listener_data, **DDS_DataReader** *reader)
*Prototype of a **DDS_DataReaderListener** (p. 1593) data_available function.*
- **typedef void(* DDS_DataReaderListener_SubscriptionMatchedCallback)** (void *listener_data, **DDS_DataReader** *reader, const struct **DDS_SubscriptionMatchedStatus** *status)
*Prototype of a **DDS_DataReaderListener** (p. 1593) subscription_match function.*
- **typedef void(* DDS_DataReaderListener_SampleLostCallback)** (void *listener_data, **DDS_DataReader** *reader, const struct **DDS_SampleLostStatus** *status)
*Prototype of a **DDS_DataReaderListener** (p. 1593) subscription_lost function.*

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. 1012) 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

- **FooDataReader * FooDataReader_narrow** (**DDS_DataReader** *reader)
*Narrow the given **DDS_DataReader** (p. 615) pointer to a **FooDataReader** (p. 2112) pointer.*
- **DDS_DataReader * FooDataReader_as_datareader** (**FooDataReader** *reader)
*Widen the given **FooDataReader** (p. 2112) pointer to a **DDS_DataReader** (p. 615) pointer.*
- **DDS_ReturnCode_t FooDataReader_read** (**FooDataReader** *self, struct **FooSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, **DDS_SampleStateMask** sample_states, **DDS_ViewStateMask** view_states, **DDS_InstanceStateMask** instance_states)
*Access a collection of data samples from the **DDS_DataReader** (p. 615).*
- **DDS_ReturnCode_t FooDataReader_take** (**FooDataReader** *self, struct **FooSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, **DDS_SampleStateMask** sample_states, **DDS_ViewStateMask** view_states, **DDS_InstanceStateMask** instance_states)
*Access a collection of data-samples from the **DDS_DataReader** (p. 615).*
- **DDS_ReturnCode_t FooDataReader_read_w_condition** (**FooDataReader** *self, struct **FooSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, **DDS_ReadCondition** *condition)
*Accesses via **FooDataReader_read** (p. 625) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).*
- **DDS_ReturnCode_t FooDataReader_take_w_condition** (**FooDataReader** *self, struct **FooSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, **DDS_ReadCondition** *condition)

Analogous to **FooDataReader_read_w_condition** (p. 630) except it accesses samples via the **FooDataReader_take** (p. 626) operation.

- **DDS_ReturnCode_t FooDataReader_read_next_sample** (**FooDataReader** *self, struct **Foo** *received_data, struct **DDS_SampleInfo** *sample_info)

*Copies the next not-previously-accessed data value from the **DDS_DataReader** (p. 615).*

- **DDS_ReturnCode_t FooDataReader_take_next_sample** (**FooDataReader** *self, struct **Foo** *received_data, struct **DDS_SampleInfo** *sample_info)

*Copies the next not-previously-accessed data value from the **DDS_DataReader** (p. 615).*

- **DDS_ReturnCode_t FooDataReader_read_instance** (**FooDataReader** *self, struct **FooSeq** *received_data, struct **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)

*Access a collection of data samples from the **DDS_DataReader** (p. 615).*

- **DDS_ReturnCode_t FooDataReader_take_instance** (**FooDataReader** *self, struct **FooSeq** *received_data, struct **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)

*Access a collection of data samples from the **DDS_DataReader** (p. 615).*

- **DDS_ReturnCode_t FooDataReader_read_instance_w_condition** (**FooDataReader** *self, struct **FooSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, const **DDS_InstanceHandle_t** *a_handle, **DDS_ReadCondition** *condition)

*<<extension>> (p. 1012) Accesses via **FooDataReader_read_instance** (p. 635) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).*

- **DDS_ReturnCode_t FooDataReader_take_instance_w_condition** (**FooDataReader** *self, struct **FooSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, const **DDS_InstanceHandle_t** *a_handle, **DDS_ReadCondition** *condition)

*<<extension>> (p. 1012) Accesses via **FooDataReader_take_instance** (p. 636) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).*

- **DDS_ReturnCode_t FooDataReader_read_next_instance** (**FooDataReader** *self, struct **FooSeq** *received_data, struct **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)

*Access a collection of data samples from the **DDS_DataReader** (p. 615).*

- **DDS_ReturnCode_t FooDataReader_take_next_instance** (**FooDataReader** *self, struct **FooSeq** *received_data, struct **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)

*Access a collection of data samples from the **DDS_DataReader** (p. 615).*

- **DDS_ReturnCode_t FooDataReader_read_next_instance_w_condition** (**FooDataReader** *self, struct **FooSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, const **DDS_InstanceHandle_t** *previous_handle, **DDS_ReadCondition** *condition)

*Accesses via **FooDataReader_read_next_instance** (p. 640) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).*

- **DDS_ReturnCode_t FooDataReader_take_next_instance_w_condition** (**FooDataReader** *self, struct **FooSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, const **DDS_InstanceHandle_t** *previous_handle, **DDS_ReadCondition** *condition)

*Accesses via **FooDataReader_take_next_instance** (p. 642) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).*

- **DDS_ReturnCode_t FooDataReader_return_loan** (**FooDataReader** *self, **FooSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq)

*Indicates to the **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615).*

- **DDS_ReturnCode_t FooDataReader_get_key_value** (FooDataReader *self, Foo *key_holder, const DDS_InstanceHandle_t *handle)
Retrieve the instance key that corresponds to an instance handle.
- **DDS_InstanceHandle_t FooDataReader_lookup_instance** (FooDataReader *self, const Foo *key_holder)
Retrieves the instance handle that corresponds to an instance key_holder.
- **DDS_ReturnCode_t FooDataReader_is_data_consistent** (FooDataReader *self, DDS_Boolean *is_data_consistent, const Foo *sample, const struct DDS_SampleInfo *sample_info)
When using Zero Copy transfer over shared memory, checks if the sample has been overwritten by the DataWriter.
- **DDS_ReturnCode_t DDS_RequestedDeadlineMissedStatus_initialize** (struct DDS_RequestedDeadlineMissedStatus *self)
Initializer for new status instances.
- **DDS_ReturnCode_t DDS_RequestedDeadlineMissedStatus_copy** (struct DDS_RequestedDeadlineMissedStatus *self, const struct DDS_RequestedDeadlineMissedStatus *source)
Copy the contents of the given status into this status.
- **DDS_ReturnCode_t DDS_RequestedDeadlineMissedStatus_finalize** (struct DDS_RequestedDeadlineMissedStatus *self)
Free any dynamic memory allocated by status instances.
- **DDS_Boolean DDS_RequestedDeadlineMissedStatus_equals** (const struct DDS_RequestedDeadlineMissedStatus *left, const struct DDS_RequestedDeadlineMissedStatus *right)
Compares two DDS_RequestedDeadlineMissedStatus (p. 1946) for equality.
- **DDS_ReturnCode_t DDS_LivelinessChangedStatus_initialize** (struct DDS_LivelinessChangedStatus *self)
Initializer for new status instances.
- **DDS_ReturnCode_t DDS_LivelinessChangedStatus_copy** (struct DDS_LivelinessChangedStatus *self, const struct DDS_LivelinessChangedStatus *source)
Copy the contents of the given status into this status.
- **DDS_ReturnCode_t DDS_LivelinessChangedStatus_finalize** (struct DDS_LivelinessChangedStatus *self)
Free any dynamic memory allocated by status instances.
- **DDS_Boolean DDS_LivelinessChangedStatus_equals** (const struct DDS_LivelinessChangedStatus *left, const struct DDS_LivelinessChangedStatus *right)
Compares two DDS_LivelinessChangedStatus (p. 1820) for equality.
- **DDS_ReturnCode_t DDS_RequestedIncompatibleQosStatus_initialize** (struct DDS_RequestedIncompatibleQosStatus *self)
Initializer for new status instances.
- **DDS_ReturnCode_t DDS_RequestedIncompatibleQosStatus_copy** (struct DDS_RequestedIncompatibleQosStatus *self, const struct DDS_RequestedIncompatibleQosStatus *source)
Copy the contents of the given status into this status.
- **DDS_ReturnCode_t DDS_RequestedIncompatibleQosStatus_finalize** (struct DDS_RequestedIncompatibleQosStatus *self)
Free any dynamic memory allocated by status instances.
- **DDS_Boolean DDS_RequestedIncompatibleQosStatus_equals** (const struct DDS_RequestedIncompatibleQosStatus *left, const struct DDS_RequestedIncompatibleQosStatus *right)
Compares two DDS_RequestedIncompatibleQosStatus (p. 1947) for equality.
- **DDS_ReturnCode_t DDS_SampleLostStatus_initialize** (struct DDS_SampleLostStatus *self)
Initializer for new status instances.
- **DDS_ReturnCode_t DDS_SampleLostStatus_copy** (struct DDS_SampleLostStatus *self, const struct DDS_SampleLostStatus *source)
Copy the contents of the given status into this status.
- **DDS_ReturnCode_t DDS_SampleLostStatus_finalize** (struct DDS_SampleLostStatus *self)

Free any dynamic memory allocated by status instances.

- **DDS_Boolean DDS_SampleLostStatus_equals** (const struct **DDS_SampleLostStatus** *left, const struct **DDS_SampleLostStatus** *right)

*Compares two **DDS_SampleLostStatus** (p. 1996) for equality.*

- **DDS_ReturnCode_t DDS_SampleRejectedStatus_initialize** (struct **DDS_SampleRejectedStatus** *self)

Initializer for new status instances.

- **DDS_ReturnCode_t DDS_SampleRejectedStatus_copy** (struct **DDS_SampleRejectedStatus** *self, const struct **DDS_SampleRejectedStatus** *source)

Copy the contents of the given status into this status.

- **DDS_ReturnCode_t DDS_SampleRejectedStatus_finalize** (struct **DDS_SampleRejectedStatus** *self)

Free any dynamic memory allocated by status instances.

- **DDS_Boolean DDS_SampleRejectedStatus_equals** (const struct **DDS_SampleRejectedStatus** *left, const struct **DDS_SampleRejectedStatus** *right)

*Compares two **DDS_SampleRejectedStatus** (p. 1997) for equality.*

- **DDS_ReturnCode_t DDS_SubscriptionMatchedStatus_initialize** (struct **DDS_SubscriptionMatchedStatus** *self)

Initializer for new status instances.

- **DDS_ReturnCode_t DDS_SubscriptionMatchedStatus_copy** (struct **DDS_SubscriptionMatchedStatus** *self, const struct **DDS_SubscriptionMatchedStatus** *source)

Copy the contents of the given status into this status.

- **DDS_ReturnCode_t DDS_SubscriptionMatchedStatus_finalize** (struct **DDS_SubscriptionMatchedStatus** *self)

Free any dynamic memory allocated by status instances.

- **DDS_Boolean DDS_SubscriptionMatchedStatus_equals** (const struct **DDS_SubscriptionMatchedStatus** *left, const struct **DDS_SubscriptionMatchedStatus** *right)

*Compares two **DDS_SubscriptionMatchedStatus** (p. 2022) for equality.*

- **DDS_ReturnCode_t DDS_DataReaderCacheStatus_initialize** (struct **DDS_DataReaderCacheStatus** *self)

Initializer for new status instances.

- **DDS_ReturnCode_t DDS_DataReaderCacheStatus_copy** (struct **DDS_DataReaderCacheStatus** *self, const struct **DDS_DataReaderCacheStatus** *source)

Copy the contents of the given status into this status.

- **DDS_ReturnCode_t DDS_DataReaderCacheStatus_finalize** (struct **DDS_DataReaderCacheStatus** *self)

Free any dynamic memory allocated by status instances.

- **DDS_Boolean DDS_DataReaderCacheStatus_equals** (const struct **DDS_DataReaderCacheStatus** *left, const struct **DDS_DataReaderCacheStatus** *right)

*Compares two **DDS_DataReaderCacheStatus** (p. 1586) for equality.*

- **DDS_ReturnCode_t DDS_DataReaderProtocolStatus_initialize** (struct **DDS_DataReaderProtocolStatus** *self)

Initializer for new status instances.

- **DDS_ReturnCode_t DDS_DataReaderProtocolStatus_copy** (struct **DDS_DataReaderProtocolStatus** *self, const struct **DDS_DataReaderProtocolStatus** *source)

Copy the contents of the given status into this status.

- **DDS_ReturnCode_t DDS_DataReaderProtocolStatus_finalize** (struct **DDS_DataReaderProtocolStatus** *self)

Free any dynamic memory allocated by status instances.

- **DDS_Boolean DDS_DataReaderProtocolStatus_equals** (const struct **DDS_DataReaderProtocolStatus** *left, const struct **DDS_DataReaderProtocolStatus** *right)

*Compares two **DDS_DataReaderProtocolStatus** (p. 1600) for equality.*

- **DDS_Boolean DDS_DataReaderQos_equals** (const struct **DDS_DataReaderQos** *self, const struct **DDS_DataReaderQos** *other)
*Compares two **DDS_DataReaderQos** (p. 1611) for equality.*
- **DDS_ReturnCode_t DDS_DataReaderQos_print** (const struct **DDS_DataReaderQos** *self)
*Prints this **DDS_DataReaderQos** (p. 1611) to stdout.*
- **DDS_ReturnCode_t DDS_DataReaderQos_to_string** (const struct **DDS_DataReaderQos** *self, char *string, **DDS_UnsignedLong** *string_size)
*Obtains a string representation of this **DDS_DataReaderQos** (p. 1611).*
- **DDS_ReturnCode_t DDS_DataReaderQos_to_string_w_params** (const struct **DDS_DataReaderQos** *self, char *string, **DDS_UnsignedLong** *string_size, const struct **DDS_DataReaderQos** *base, const struct **DDS_QosPrintFormat** *format)
*Obtains a string representation of this **DDS_DataReaderQos** (p. 1611).*
- **DDS_ReturnCode_t DDS_DataReaderQos_initialize** (struct **DDS_DataReaderQos** *self)
Initializer for new QoS instances.
- **DDS_ReturnCode_t DDS_DataReaderQos_copy** (struct **DDS_DataReaderQos** *self, const struct **DDS_DataReaderQos** *source)
Copy the contents of the given QoS into this QoS.
- **DDS_ReturnCode_t DDS_DataReaderQos_finalize** (struct **DDS_DataReaderQos** *self)
*Free any dynamic memory allocated by the policies in this **DDS_DataReaderQos** (p. 1611).*
- **DDS_Entity * DDS_DataReader_as_entity** (**DDS_DataReader** *dataReader)
*Accesses the **DDS_DataReader** (p. 615)'s supertype instance.*
- **DDS_ReadCondition * DDS_DataReader_create_readcondition** (**DDS_DataReader** *self, **DDS_SampleStateMask** sample_states, **DDS_ViewStateMask** view_states, **DDS_InstanceStateMask** instance_states)
*Creates a **DDS_ReadCondition** (p. 694).*
- **DDS_ReadCondition * DDS_DataReader_create_readcondition_w_params** (**DDS_DataReader** *self, const struct **DDS_ReadConditionParams** *params)
*<<extension>> (p. 1012) Creates a **DDS_ReadCondition** (p. 694) with parameters.*
- **DDS_QueryCondition * DDS_DataReader_create_querycondition** (**DDS_DataReader** *self, **DDS_SampleStateMask** sample_states, **DDS_ViewStateMask** view_states, **DDS_InstanceStateMask** instance_states, const char *query_expression, const struct **DDS_StringSeq** *query_parameters)
*Creates a **DDS_QueryCondition** (p. 699).*
- **DDS_QueryCondition * DDS_DataReader_create_querycondition_w_params** (**DDS_DataReader** *self, const struct **DDS_QueryConditionParams** *params)
*<<extension>> (p. 1012) Creates a **DDS_QueryCondition** (p. 699) with parameters.*
- **DDS_ReturnCode_t DDS_DataReader_delete_readcondition** (**DDS_DataReader** *self, **DDS_ReadCondition** *condition)
*Deletes a **DDS_ReadCondition** (p. 694) or **DDS_QueryCondition** (p. 699) attached to the **DDS_DataReader** (p. 615).*
- **DDS_ReturnCode_t DDS_DataReader_delete_contained_entities** (**DDS_DataReader** *self)
*Deletes all the entities that were created by means of the "create" operations on the **DDS_DataReader** (p. 615).*
- **DDS_ReturnCode_t DDS_DataReader_wait_for_historical_data** (**DDS_DataReader** *self, const struct **DDS_Duration_t** *max_wait)
*Waits until all "historical" data is received for **DDS_DataReader** (p. 615) entities that have a non-VOLATILE Durability Qos kind.*
- **DDS_ReturnCode_t DDS_DataReader_acknowledge_sample_w_response** (**DDS_DataReader** *self, const struct **DDS_SampleInfo** *sample_info, const struct **DDS_AckResponseData_t** *response_data)
<<extension>> (p. 1012) Acknowledges a single sample explicitly.
- **DDS_ReturnCode_t DDS_DataReader_acknowledge_all_w_response** (**DDS_DataReader** *self, const struct **DDS_AckResponseData_t** *response_data)
<<extension>> (p. 1012) Acknowledges all previously accessed samples.

- **DDS_ReturnCode_t DDS_DataReader_acknowledge_sample** (DDS_DataReader *self, const struct DDS_↵_SampleInfo *sample_info)
 <<extension>> (p. 1012) Acknowledges a single sample explicitly.
- **DDS_ReturnCode_t DDS_DataReader_acknowledge_all** (DDS_DataReader *self)
 <<extension>> (p. 1012) Acknowledges all previously accessed samples.
- **DDS_ReturnCode_t DDS_DataReader_get_matched_publications** (DDS_DataReader *self, struct DDS_↵_InstanceHandleSeq *publication_handles)
 Retrieves the list of publications currently "associated" with this **DDS_DataReader** (p. 615).
- **DDS_ReturnCode_t DDS_DataReader_is_matched_publication_alive** (DDS_DataReader *self, DDS_↵_Boolean *is_alive, const DDS_InstanceHandle_t *publication_handle)
 Check if a publication currently matched with a DataReader is alive.
- **DDS_ReturnCode_t DDS_DataReader_get_matched_publication_data** (DDS_DataReader *self, struct DDS_PublicationBuiltinTopicData *publication_data, const DDS_InstanceHandle_t *publication_handle)
 Retrieves the information on a publication that is currently "associated" with the **DDS_DataReader** (p. 615).
- **DDS_ReturnCode_t DDS_DataReader_get_matched_publication_participant_data** (DDS_Data_↵Reader *self, struct DDS_ParticipantBuiltinTopicData *participant_data, const DDS_InstanceHandle_t *publication_handle)
 <<extension>> (p. 1012) Retrieves the information on the discovered **DDS_DomainParticipant** (p. 77) associated with the publication that is currently matching with the **DDS_DataReader** (p. 615).
- **DDS_TopicDescription * DDS_DataReader_get_topicdescription** (DDS_DataReader *self)
 Returns the **DDS_TopicDescription** (p. 175) associated with the **DDS_DataReader** (p. 615).
- **DDS_Subscriber * DDS_DataReader_get_subscriber** (DDS_DataReader *self)
 Returns the **DDS_Subscriber** (p. 572) to which the **DDS_DataReader** (p. 615) belongs.
- **DDS_ReturnCode_t DDS_DataReader_get_sample_rejected_status** (DDS_DataReader *self, struct DDS_SampleRejectedStatus *status)
 Accesses the **DDS_SAMPLE_REJECTED_STATUS** (p. 1230) communication status.
- **DDS_ReturnCode_t DDS_DataReader_get_liveliness_changed_status** (DDS_DataReader *self, struct DDS_LivelinessChangedStatus *status)
 Accesses the **DDS_LIVELINESS_CHANGED_STATUS** (p. 1231) communication status.
- **DDS_ReturnCode_t DDS_DataReader_get_requested_deadline_missed_status** (DDS_DataReader *self, struct DDS_RequestedDeadlineMissedStatus *status)
 Accesses the **DDS_REQUESTED_DEADLINE_MISSED_STATUS** (p. 1229) communication status.
- **DDS_ReturnCode_t DDS_DataReader_get_requested_incompatible_qos_status** (DDS_DataReader *self, struct DDS_RequestedIncompatibleQosStatus *status)
 Accesses the **DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 1229) communication status.
- **DDS_ReturnCode_t DDS_DataReader_get_subscription_matched_status** (DDS_DataReader *self, struct DDS_SubscriptionMatchedStatus *status)
 Accesses the **DDS_SUBSCRIPTION_MATCHED_STATUS** (p. 1232) communication status.
- **DDS_ReturnCode_t DDS_DataReader_get_sample_lost_status** (DDS_DataReader *self, struct DDS_↵_SampleLostStatus *status)
 Accesses the **DDS_SAMPLE_LOST_STATUS** (p. 1230) communication status.
- **DDS_ReturnCode_t DDS_DataReader_get_datareader_cache_status** (DDS_DataReader *self, struct DDS_DataReaderCacheStatus *status)
 <<extension>> (p. 1012) Gets the datareader cache status for this reader.
- **DDS_ReturnCode_t DDS_DataReader_get_datareader_protocol_status** (DDS_DataReader *self, struct DDS_DataReaderProtocolStatus *status)
 <<extension>> (p. 1012) Gets the datareader protocol status for this reader.
- **DDS_ReturnCode_t DDS_DataReader_get_matched_publication_datareader_protocol_status** (DDS_↵_DataReader *self, struct DDS_DataReaderProtocolStatus *status, const DDS_InstanceHandle_↵_t *publication_handle)

<<**extension**>> (p. 1012) Gets the datareader protocol status for this reader, per matched publication identified by the `publication_handle`.

- **DDS_ReturnCode_t DDS_DataReader_set_qos** (**DDS_DataReader** *self, const struct **DDS_DataReaderQos** *qos)

Sets the reader QoS.

- **DDS_ReturnCode_t DDS_DataReader_set_qos_with_profile** (**DDS_DataReader** *self, const char *library_name, const char *profile_name)

<<**extension**>> (p. 1012) Changes the QoS of this reader using the input XML QoS profile.

- **DDS_ReturnCode_t DDS_DataReader_get_qos** (**DDS_DataReader** *self, struct **DDS_DataReaderQos** *qos)

Gets the reader QoS.

- **DDS_ReturnCode_t DDS_DataReader_set_property** (**DDS_DataReader** *self, const char *property_name, const char *value, **DDS_Boolean** propagate)

Set the value for a property that applies to a DataReader.

- **DDS_ReturnCode_t DDS_DataReader_set_listener** (**DDS_DataReader** *self, const struct **DDS_DataReaderListener** *l, **DDS_StatusMask** mask)

Sets the reader listener.

- struct **DDS_DataReaderListener** **DDS_DataReader_get_listener** (**DDS_DataReader** *self)

Gets the reader listener.

- **DDS_ReturnCode_t DDS_DataReader_get_listenerX** (**DDS_DataReader** *self, struct **DDS_DataReaderListener** *listener)

<<**extension**>> (p. 1012) Gets the reader listener.

- **DDS_TopicQuery * DDS_DataReader_create_topic_query** (**DDS_DataReader** *self, const struct **DDS_TopicQuerySelection** *selection)

*Creates a **DDS_TopicQuery** (p. 707).*

- **DDS_ReturnCode_t DDS_DataReader_delete_topic_query** (**DDS_DataReader** *self, **DDS_TopicQuery** *query)

*Deletes a **DDS_TopicQuery** (p. 707).*

- **DDS_TopicQuery * DDS_DataReader_lookup_topic_query** (**DDS_DataReader** *self, const struct **DDS_GUID_t** *guid)

*Retrieves an existing **DDS_TopicQuery** (p. 707).*

- **DDS_ReturnCode_t DDS_DataReader_take_discovery_snapshot** (**DDS_DataReader** *self, const char *file_name)

Take a snapshot of the compatible and incompatible remote writers matched by a local reader.

5.21.1 Detailed Description

DDS_DataReader (p. 615) entity and associated elements

5.21.2 Macro Definition Documentation

5.21.2.1 DDS_RequestedDeadlineMissedStatus_INITIALIZER

```
#define DDS_RequestedDeadlineMissedStatus_INITIALIZER
```

Initializer for new status instances.

New **DDS_RequestedDeadlineMissedStatus** (p. 1946) instances that are stored on the stack should be initialized with this value before they are passed to any methods. This step ensures that those fields that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a status structure is to initialize it on the stack at the time of its creation. **DDS_InconsistentTopicStatus_finalize** (p. 188) should be called to free the contained fields that use dynamic memory:

```
struct DDS_RequestedDeadlineMissedStatus myStatus = DDS_RequestedDeadlineMissedStatus_INITIALIZER;
```

```
...
```

```
DDS_RequestedDeadlineMissedStatus_finalize(&myStatus);
```

See also

DDS_RequestedDeadlineMissedStatus_initialize (p. 650)

DDS_RequestedDeadlineMissedStatus_finalize (p. 651)

5.21.2.2 DDS_LivelinessChangedStatus_INITIALIZER

```
#define DDS_LivelinessChangedStatus_INITIALIZER
```

Initializer for new status instances.

New **DDS_LivelinessChangedStatus** (p. 1820) instances that are stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those fields that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a status structure is to initialize it on the stack at the time of its creation. **DDS_LivelinessChangedStatus_finalize** (p. 652) should be called to free the contained fields that use dynamic memory:

```
struct DDS_LivelinessChangedStatus myStatus = DDS_LivelinessChangedStatus_INITIALIZER;
```

```
...
```

```
DDS_LivelinessChangedStatus_finalize(&myStatus);
```

See also

DDS_LivelinessChangedStatus_initialize (p. 651)

DDS_LivelinessChangedStatus_finalize (p. 652)

5.21.2.3 DDS_RequestedIncompatibleQosStatus_INITIALIZER

```
#define DDS_RequestedIncompatibleQosStatus_INITIALIZER
```

Initializer for new status instances.

New **DDS_RequestedIncompatibleQosStatus** (p. 1947) instances that are stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those fields that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a status structure is to initialize it on the stack at the time of its creation. **DDS_RequestedIncompatibleQosStatus_finalize** (p. 654) should be called to free the contained fields that use dynamic memory:

```
struct DDS_RequestedIncompatibleQosStatus myStatus = DDS_RequestedIncompatibleQosStatus_INITIALIZER;
```

```
...
```

```
DDS_RequestedIncompatibleQosStatus_finalize(&myStatus);
```

See also

DDS_RequestedIncompatibleQosStatus_initialize (p. 653)

DDS_RequestedIncompatibleQosStatus_finalize (p. 654)

5.21.2.4 DDS_SampleLostStatus_INITIALIZER

```
#define DDS_SampleLostStatus_INITIALIZER
```

Initializer for new status instances.

New **DDS_SampleLostStatus** (p. 1996) instances that are stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those fields that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a status structure is to initialize it on the stack at the time of its creation. **DDS_SampleLostStatus_finalize** (p. 656) should be called to free the contained fields that use dynamic memory:

```
struct DDS_SampleLostStatus myStatus = DDS_SampleLostStatus_INITIALIZER;
```

```
...
```

```
DDS_SampleLostStatus_finalize(&myStatus);
```

See also

DDS_SampleLostStatus_initialize (p. 655)

DDS_SampleLostStatus_finalize (p. 656)

5.21.2.5 DDS_SampleRejectedStatus_INITIALIZER

```
#define DDS_SampleRejectedStatus_INITIALIZER
```

Initializer for new status instances.

New **DDS_SampleRejectedStatus** (p. 1997) instances that are stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those fields that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a status structure is to initialize it on the stack at the time of its creation. **DDS_SampleRejectedStatus_finalize** (p. 658) should be called to free the contained fields that use dynamic memory:

```
struct DDS_SampleRejectedStatus myStatus = DDS_SampleRejectedStatus_INITIALIZER;
```

```
...
```

```
DDS_SampleRejectedStatus_finalize(&myStatus);
```

See also

DDS_SampleRejectedStatus_initialize (p. 657)

DDS_SampleRejectedStatus_finalize (p. 658)

5.21.2.6 DDS_SubscriptionMatchedStatus_INITIALIZER

```
#define DDS_SubscriptionMatchedStatus_INITIALIZER
```

Initializer for new status instances.

New **DDS_SubscriptionMatchedStatus** (p. 2022) instances that are stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those fields that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a status structure is to initialize it on the stack at the time of its creation. **DDS_SubscriptionMatchedStatus_finalize** (p. 660) should be called to free the contained fields that use dynamic memory:

```
struct DDS_SubscriptionMatchedStatus myStatus = DDS_SubscriptionMatchedStatus_INITIALIZER;
```

```
...
```

```
DDS_SubscriptionMatchedStatus_finalize(&myStatus);
```

See also

DDS_SubscriptionMatchedStatus_initialize (p. 659)

DDS_SubscriptionMatchedStatus_finalize (p. 660)

5.21.2.7 DDS_DataReaderCacheStatus_INITIALIZER

```
#define DDS_DataReaderCacheStatus_INITIALIZER
```

Initializer for new status instances.

New **DDS_DataReaderCacheStatus** (p. 1586) instances stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those fields that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a status structure is to initialize it on the stack at the time of its creation. **DDS_DataReaderCacheStatus_finalize** (p. 662) should be called to free the contained fields that use dynamic memory:

```
struct DDS_DataReaderCacheStatus myStatus = DDS_DataReaderCacheStatus_INITIALIZER;
```

```
DDS_DataReader_get_datareader_cache_status(myDataReader, &myStatus);
```

```
DDS_DataReaderCacheStatus_finalize(&myStatus);
```

See also

DDS_DataReader_get_datareader_cache_status (p. 685)

DDS_DataReaderCacheStatus_finalize (p. 662)

5.21.2.8 DDS_DataReaderProtocolStatus_INITIALIZER

```
#define DDS_DataReaderProtocolStatus_INITIALIZER
```

Initializer for new status instances.

New **DDS_DataReaderProtocolStatus** (p. 1600) instances stored on the stack should be initialized with this value before they are passed to any function. This step ensures that those fields that use dynamic memory are properly initialized. This does not allocate memory.

The simplest way to create a status structure is to initialize it on the stack at the time of its creation. **DDS_DataReaderProtocolStatus_finalize** (p. 664) should be called to free the contained fields that use dynamic memory:

```
struct DDS_DataReaderProtocolStatus myStatus = DDS_DataReaderProtocolStatus_INITIALIZER;
```

```
DDS_DataReader_get_datareader_protocol_status(myDataReader, &myStatus);
```

```
DDS_DataReaderProtocolStatus_finalize(&myStatus);
```

See also

DDS_DataReader_get_datareader_protocol_status (p. 686)

DDS_DataReaderProtocolStatus_finalize (p. 664)

5.21.2.9 DDS_DataReaderQos_INITIALIZER

```
#define DDS_DataReaderQos_INITIALIZER
```

Initializer for new QoS instances.

New **DDS_DataReaderQos** (p. 1611) instances that are stored in the stack should be initialized with this value before they are passed to any function. This step ensures that those contained QoS policies that use dynamic memory are properly initialized.

The simplest way to create a new QoS structure is to initialize it on the stack at the time of its creation:

```
struct DDS_DataReaderQos myQos = DDS_DataReaderQos_INITIALIZER;
```

Note that the above assignment is not a substitute for calling **DDS_Subscriber_get_default_datareader_qos** (p. 579) or **DDS_DataReader_get_qos** (p. 688); one of those should be called subsequently to setting the QoS of a new or existing entity:

```
struct DDS_DataReaderQos myQos = DDS_DataReaderQos_INITIALIZER;
```

```
DDS_Subscriber_get_default_datareader_qos(mySub, &myQos);
```

```
DDS_DataReader_get_qos(myReader, &myQos);
```

See also

DDS_DataReaderQos_initialize (p. 669)

DDS_Subscriber_get_default_datareader_qos (p. 579)

DDS_DataReaderQos_finalize (p. 671)

5.21.2.10 DDS_DataReaderListener_INITIALIZER

```
#define DDS_DataReaderListener_INITIALIZER
```

Initializer for new **DDS_DataReaderListener** (p. 1593).

All the new instances allocated in the stack should be initialized to this value. No memory is allocated.

Examples

HelloWorld_subscriber.c.

5.21.3 Typedef Documentation

5.21.3.1 DDS_DataReader

```
typedef struct DDS_DataReaderImpl DDS_DataReader
```

<<**interface**>> (p. 1013) 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 **DDS_Subscriber** (p. 572).

QoS:

DDS_DataReaderQos (p. 1611)

Status:

DDS_DATA_AVAILABLE_STATUS (p. 1230);
DDS_LIVELINESS_CHANGED_STATUS (p. 1231), **DDS_LivelinessChangedStatus** (p. 1820);
DDS_REQUESTED_DEADLINE_MISSED_STATUS (p. 1229), **DDS_RequestedDeadlineMissedStatus** (p. 1946);
DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS (p. 1229), **DDS_RequestedIncompatibleQosStatus** (p. 1947);
DDS_SAMPLE_LOST_STATUS (p. 1230), **DDS_SampleLostStatus** (p. 1996);
DDS_SAMPLE_REJECTED_STATUS (p. 1230), **DDS_SampleRejectedStatus** (p. 1997);
DDS_SUBSCRIPTION_MATCHED_STATUS (p. 1232), **DDS_SubscriptionMatchedStatus** (p. 2022);

Listener:

DDS_DataReaderListener (p. 1593)

A **DDS_DataReader** (p. 615) refers to exactly one **DDS_TopicDescription** (p. 175) (either a **DDS_Topic** (p. 175), a **DDS_ContentFilteredTopic** (p. 177) or a **DDS_MultiTopic** (p. 185)) 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*.

DDS_DataReader (p. 615) is an abstract class. It must be specialised for each particular application data-type (see **USER_DATA** (p. 1343)). The additional methods or functions that must be defined in the auto-generated class for a hypothetical application type **Foo** (p. 2109) are specified in the generic type **FooDataReader** (p. 2112).

The following operations may be called even if the **DDS_DataReader** (p. 615) is not enabled. Other operations will fail with the value **DDS_RETCODE_NOT_ENABLED** (p. 1222) if called on a disabled **DDS_DataReader** (p. 615):

- The base-class operations **DDS_DataReader_set_qos** (p. 687), **DDS_DataReader_get_qos** (p. 688), **DDS_DataReader_set_listener** (p. 690), **DDS_DataReader_get_listener** (p. 690), **DDS_Entity_enable** (p. 1363), **DDS_Entity_get_statuscondition** (p. 1365), **DDS_Entity_get_status_changes** (p. 1365),
- **DDS_DataReader_get_liveliness_changed_status** (p. 682), **DDS_DataReader_get_requested_deadline_missed_status** (p. 682), **DDS_DataReader_get_requested_incompatible_qos_status** (p. 683), **DDS_DataReader_get_sample_lost_status** (p. 685), **DDS_DataReader_get_sample_rejected_status** (p. 682), **DDS_DataReader_get_subscription_matched_status** (p. 683)

All sample-accessing operations, namely: **FooDataReader_read** (p. 625), **FooDataReader_take** (p. 626), **FooDataReader_read_w_condition** (p. 630), and **FooDataReader_take_w_condition** (p. 632) may fail with the error **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) as described in **DDS_Subscriber_begin_access** (p. 587).

See also

Operations Allowed in Listener Callbacks (p. 1818)

Access to data samples (p. 567)

5.21.3.2 DDS_DataReaderListener_RequestedDeadlineMissedCallback

```
typedef void(* DDS_DataReaderListener_RequestedDeadlineMissedCallback) (void *listener_data, DDS_↵  
_DataReader *reader, const struct DDS_RequestedDeadlineMissedStatus *status)
```

Prototype of a **DDS_DataReaderListener** (p. 1593) requested_deadline_missed function.

5.21.3.3 DDS_DataReaderListener_LivelinessChangedCallback

```
typedef void(* DDS_DataReaderListener_LivelinessChangedCallback) (void *listener_data, DDS_Data↵  
Reader *reader, const struct DDS_LivelinessChangedStatus *status)
```

Prototype of a **DDS_DataReaderListener** (p. 1593) liveliness_changed function.

5.21.3.4 DDS_DataReaderListener_RequestedIncompatibleQosCallback

```
typedef void(* DDS_DataReaderListener_RequestedIncompatibleQosCallback) (void *listener_data,  
DDS_DataReader *reader, const struct DDS_RequestedIncompatibleQosStatus *status)
```

Prototype of a **DDS_DataReaderListener** (p. 1593) requested_incompatible_QoS function.

5.21.3.5 DDS_DataReaderListener_SampleRejectedCallback

```
typedef void(* DDS_DataReaderListener_SampleRejectedCallback) (void *listener_data, DDS_Data↵  
Reader *reader, const struct DDS_SampleRejectedStatus *status)
```

Prototype of a **DDS_DataReaderListener** (p. 1593) sample_rejected function.

5.21.3.6 DDS_DataReaderListener_DataAvailableCallback

```
typedef void(* DDS_DataReaderListener_DataAvailableCallback) (void *listener_data, DDS_DataReader  
*reader)
```

Prototype of a **DDS_DataReaderListener** (p. 1593) data_available function.

5.21.3.7 DDS_DataReaderListener_SubscriptionMatchedCallback

```
typedef void(* DDS_DataReaderListener_SubscriptionMatchedCallback) (void *listener_data, DDS_↵
DataReader *reader, const struct DDS_SubscriptionMatchedStatus *status)
```

Prototype of a **DDS_DataReaderListener** (p. 1593) subscription_match function.

5.21.3.8 DDS_DataReaderListener_SampleLostCallback

```
typedef void(* DDS_DataReaderListener_SampleLostCallback) (void *listener_data, DDS_DataReader
*reader, const struct DDS_SampleLostStatus *status)
```

Prototype of a **DDS_DataReaderListener** (p. 1593) subscription_lost function.

5.21.4 Enumeration Type Documentation

5.21.4.1 DDS_SampleLostStatusKind

```
enum DDS_SampleLostStatusKind
```

<<*extension*>> (p. 1012) 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 DDS_DataWriter (p. 485) removed the sample before being received by the DDS_DataReader (p. 615). 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. 1951)) was reached. This constant is an extension to the DDS standard. See also DDS_ResourceLimitsQosPolicy (p. 1949)

Enumerator

DDS_LOST_BY_REMOTE_WRITERS_PER_INSTANCE_LIMIT ↩	A resource limit on the number of remote writers for a single instance from which a DDS_DataReader (p. 615) may read (DDS_DataReaderResourceLimitsQos ↩ Policy::max_remote_writers_per_instance (p. 1622)) was reached. This constant is an extension to the DDS standard. See also DDS_DataReaderResourceLimitsQosPolicy (p. 1619)
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 DDS_DataWriter (p. 485) using DDS_KEEP_LAST_HISTORY_QOS (p. 1293) 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 DDS_DataReader (p. 615). 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. 1894) 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. 1985) 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 DDS_DataReader (p. 615) queue because resource limits are exceeded. For example, if DDS_ResourceLimitsQosPolicy::max_samples_per_instance (p. 1952) 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. 1951), DDS_ResourceLimitsQosPolicy::max_samples_per_instance (p. 1952), DDS_ResourceLimitsQosPolicy::max_instances (p. 1951), and DDS_DataReaderResourceLimitsQosPolicy::max_samples_per_remote_writer (p. 1622). 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. 1323): a resource limit on the number of samples from a given remote writer that a DDS_DataReader (p. 615) may store (DDS_DataReaderResourceLimitsQosPolicy↔::max_samples_per_remote_writer (p. 1622)) was reached. When using DDS_RELIABLE_RELIABILITY_QOS (p. 1323), reaching DDS_DataReaderResourceLimitsQosPolicy↔::max_samples_per_remote_writer (p. 1622) will trigger a rejection, not a loss, with reason DDS_REJECTED_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT (p. 623). This constant is an extension to the DDS standard. See also DDS_DataReaderResourceLimitsQosPolicy (p. 1619)
DDS_LOST_BY_VIRTUAL_WRITERS_LIMIT	A resource limit on the number of virtual writers from which a DDS_DataReader (p. 615) may read (DDS_DataReaderResourceLimitsQosPolicy↔::max_remote_virtual_writers (p. 1630)) was reached. This constant is an extension to the DDS standard. See also DDS_DataReaderResourceLimitsQosPolicy (p. 1619)
DDS_LOST_BY_REMOTE_WRITERS_PER_SAMPLE_LIMIT↔	A resource limit on the number of remote writers per sample (DDS_DataReaderResourceLimitsQosPolicy↔::max_remote_writers_per_sample (p. 1632)) was reached. This constant is an extension to the DDS standard. See also DDS_DataReaderResourceLimitsQosPolicy (p. 1619)
DDS_LOST_BY_AVAILABILITY_WAITING_TIME	DDS_AvailabilityQosPolicy↔::max_data_availability_waiting_time (p. 1550) expired. This constant is an extension to the DDS standard. See also DDS_AvailabilityQosPolicy (p. 1547)

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 DDS_DataReader (p. 615) may store was reached. (This field is currently not used.) This constant is an extension to the DDS standard. See also DDS_DataReaderResourceLimitsQosPolicy (p. 1619)
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. 1619)
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. 1323): A received sample was lost because it could not be decoded. When using DDS_RELIABLE_RELIABILITY_QOS (p. 1323), the sample will be rejected, not lost, with reason DDS_REJECTED_BY_DECODE_FAILURE (p. 624). This constant is an extension to the DDS standard.
DDS_LOST_BY_SAMPLES_PER_INSTANCE_LIMIT	When using DDS_BEST_EFFORT_RELIABILITY_QOS (p. 1323): A resource limit on the number of samples per instance (DDS_ResourceLimitsQosPolicy::max_samples_↵ per_instance (p. 1952)) was reached. When using DDS_RELIABLE_RELIABILITY_QOS (p. 1323), reaching DDS_ResourceLimitsQosPolicy::max_↵ samples_per_instance (p. 1952) will trigger a rejection, not a loss, with reason DDS_REJECTED_BY_↵ SAMPLES_PER_INSTANCE_LIMIT (p. 623). This constant is an extension to the DDS standard. See also DDS_ResourceLimitsQosPolicy (p. 1949)

Enumerator

DDS_LOST_BY_SAMPLES_LIMIT	When using DDS_BEST_EFFORT_RELIABILITY_QOS (p. 1323): A resource limit on the number of samples (DDS_ResourceLimitsQosPolicy::max_samples (p. 1951)) was reached. When using DDS_RELIABLE_RELIABILITY_QOS (p. 1323), reaching DDS_ResourceLimitsQosPolicy::max_samples (p. 1951) will trigger a rejection, not a loss, with reason DDS_REJECTED_BY_SAMPLES_LIMIT (p. 622). This constant is an extension to the DDS standard. See also DDS_ResourceLimitsQosPolicy (p. 1949)

5.21.4.2 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	Connex DDS does not reject samples based on instance limits (DDS_ResourceLimitsQosPolicy::max_instances (p. 1951)), so this value will never be used. See also DDS_LOST_BY_INSTANCES_LIMIT (p. 618) 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. 1323): A resource limit on the number of samples (DDS_ResourceLimitsQosPolicy::max_samples (p. 1951)) was reached. When using DDS_BEST_EFFORT_RELIABILITY_QOS (p. 1323), reaching DDS_ResourceLimitsQosPolicy::max_samples (p. 1951) will trigger a loss, not a rejection, with reason DDS_LOST_BY_SAMPLES_LIMIT (p. 622). See also DDS_ResourceLimitsQosPolicy (p. 1949)
	Generated by Doxygen

Enumerator

DDS_REJECTED_BY_SAMPLES_PER_INSTANCE_↔ _LIMIT	When using DDS_RELIABLE_RELIABILITY_QOS (p. 1323): A resource limit on the number of samples per instance (DDS_ResourceLimitsQosPolicy::max_↔ samples_per_instance (p. 1952)) was reached. When using DDS_BEST_EFFORT_RELIABILITY_QOS (p. 1323), reaching DDS_ResourceLimitsQosPolicy_↔ ::max_samples_per_instance (p. 1952) will trigger a loss, not a rejection, with reason DDS_LOST_BY_SAMPLES_PER_INSTANCE_LIMIT (p. 621). See also ResourceLimitsQosPolicy
DDS_REJECTED_BY_SAMPLES_PER_REMOTE_↔ WRITER_LIMIT	When using DDS_RELIABLE_RELIABILITY_QOS (p. 1323): a resource limit on the number of samples from a given remote writer that a DDS_DataReader (p. 615) may store (DDS_DataReaderResource_↔ LimitsQosPolicy::max_samples_per_remote_writer (p. 1622)) was reached. When using DDS_BEST_EFFORT_RELIABILITY_QOS (p. 1323), reaching DDS_DataReaderResourceLimitsQos_↔ Policy::max_samples_per_remote_writer (p. 1622) will trigger a loss, not a rejection, with reason DDS_LOST_BY_SAMPLES_PER_REMOTE_↔ WRITER_LIMIT (p. 620). This constant is an extension to the DDS standard. See also DDS_DataReaderResourceLimitsQosPolicy (p. 1619)
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 DDS_DataReader (p. 615) may store was reached. (This field is currently not used.) This constant is an extension to the DDS standard. See also DDS_DataReaderResourceLimitsQosPolicy (p. 1619)

Enumerator

DDS_REJECTED_BY_DECODE_FAILURE	When using DDS_RELIABLE_RELIABILITY_QOS (p. 1323): A received sample was rejected because it could not be decoded. When using DDS_BEST_EFFORT_RELIABILITY_QOS (p. 1323), the sample will be lost, not rejected, with reason DDS_LOST_BY_DECODE_FAILURE (p. 621). If a sample was rejected for this reason and the DDS_DataWriter (p. 485) set DDS_DataWriter↔ProtocolQosPolicy::disable_inline_keyhash (p. 1651) to DDS_BOOLEAN_TRUE (p. 1201), then DDS_SampleRejectedStatus::last_instance_handle (p. 1998) may not be correct if the sample was encrypted. This constant is an extension to the DDS standard.

5.21.5 Function Documentation

5.21.5.1 FooDataReader_narrow()

```

FooDataReader * FooDataReader_narrow (
    DDS_DataReader * reader )

```

Narrow the given **DDS_DataReader** (p. 615) pointer to a **FooDataReader** (p. 2112) pointer.

Parameters

<i>reader</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.21.5.2 FooDataReader_as_datareader()

```

DDS_DataReader * FooDataReader_as_datareader (
    FooDataReader * reader )

```

Widen the given **FooDataReader** (p. 2112) pointer to a **DDS_DataReader** (p. 615) pointer.

Parameters

<i>reader</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.21.5.3 FooDataReader_read()

```
DDS_ReturnCode_t FooDataReader_read (
    FooDataReader * self,
    struct FooSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    DDS_SampleStateMask sample_states,
    DDS_ViewStateMask view_states,
    DDS_InstanceStateMask instance_states )
```

Access a collection of data samples from the **DDS_DataReader** (p. 615).

This operation offers the same functionality and API as **FooDataReader_take** (p. 626) except that the samples returned remain in the **DDS_DataReader** (p. 615) such that they can be retrieved again by means of a read or take operation.

Please refer to the documentation of **FooDataReader_take()** (p. 626) 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. 711). 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. 713). 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 **DDS_DataReader** (p. 615) by calling the **FooDataReader_return_loan** (p. 646) 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. 1985) objects after the call to **FooDataReader_return_loan** (p. 646). 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. 626) call. However, failure to return the loan will eventually deplete the **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615) is specified by the **DDS_ResourceLimitsQosPolicy** (p. 1949) and the **DDS_DataReader_←ResourceLimitsQosPolicy** (p. 1619).

Important: If the samples "returned" by this function are loaned from RTI Connext (see **FooDataReader_take** (p. 626) 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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>received_data</i>	<< <i>inout</i> >> (p. 1013) User data type-specific FooSeq (p. 2113) object where the received data samples will be returned. Must be a valid non-NULL FooSeq (p. 2113). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>info_seq</i>	<< <i>inout</i> >> (p. 1013) A DDS_SampleInfoSeq (p. 1996) object where the received sample info will be returned. Must be a valid non-NULL DDS_SampleInfoSeq (p. 1996). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>max_samples</i>	<< <i>in</i> >> (p. 1013) The maximum number of samples to be returned. If the special value
Generated by Doxygen	DDS_LENGTH_UNLIMITED (p. 1326) is provided, as many samples will be returned as are available, up to the limits described in the documentation for FooDataReader_take() (p. 626).
<i>sample_states</i>	<< <i>in</i> >> (p. 1013) Data samples matching one of these <code>sample_states</code> are returned.
<i>view_states</i>	<< <i>in</i> >> (p. 1013) Data samples matching one of these <code>view_state</code> are returned.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222), **DDS_RETCODE_NO_DATA** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

FooDataReader_take (p. 626)
FooDataReader_read_w_condition (p. 630),
FooDataReader_take_w_condition (p. 632)
DDS_LENGTH_UNLIMITED (p. 1326)

5.21.5.4 FooDataReader_take()

```
DDS_ReturnCode_t FooDataReader_take (
    FooDataReader * self,
    struct FooSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    DDS_SampleStateMask sample_states,
    DDS_ViewStateMask view_states,
    DDS_InstanceStateMask instance_states )
```

Access a collection of data-samples from the **DDS_DataReader** (p. 615).

The operation will return the list of samples received by the **DDS_DataReader** (p. 615) since the last **FooDataReader_take** (p. 626) operation that matches the specified **DDS_SampleStateMask** (p. 710), **DDS_ViewStateMask** (p. 712) and **DDS_InstanceStateMask** (p. 714).

This operation may fail with **DDS_RETCODE_ERROR** (p. 1222) if the **DDS_DataReaderResourceLimitsQosPolicy::max_outstanding_reads** (p. 1625) 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. 1805), **DDS_ResourceLimitsQosPolicy** (p. 1949), **DDS_DataReaderResourceLimitsQosPolicy** (p. 1619) and the characteristics of the data-type that is associated with the **DDS_DataReader** (p. 615):

- In the case where the **DDS_HistoryQosPolicy::kind** (p. 1807) is **DDS_KEEP_LAST_HISTORY_QOS** (p. 1293), the call will return at most **DDS_HistoryQosPolicy::depth** (p. 1807) samples for each ALIVE instance and (**DDS_HistoryQosPolicy::depth** (p. 1807) +1) samples for each NOT_ALIVE instance. The extra sample is an invalid sample (**DDS_SampleInfo::valid_data** (p. 1992) 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. 1951), and by **DDS_DataReaderResourceLimitsQosPolicy::max_samples_per_read** (p. 1625).

- For multiple instances, the number of samples returned is additionally limited by the product (**DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1952) – **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1951))
- If **DDS_DataReaderResourceLimitsQosPolicy::max_infos** (p. 1623) is limited, the number of samples returned may also be limited if insufficient **DDS_SampleInfo** (p. 1985) 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. 1985) 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 **DDS_Topic** (p. 175) associated with the **DDS_DataReader** (p. 615) is bound to a data-type that has no key definition, then there will be at most one instance in the **DDS_DataReader** (p. 615). 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. 713). It will not affect the `instance_state` of the sample's instance.

After **FooDataReader_take** (p. 626) 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. 626) is called, the samples returned in the `received_data` and the corresponding `info_seq` are 'loaned' to the application from buffers provided by the **DDS_DataReader** (p. 615). 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 **DDS_DataReader** (p. 615) by calling the **FooDataReader_return_loan** (p. 646) 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. 1985) objects after the call to **FooDataReader_return_loan** (p. 646). 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. 646) at some point, you do *not* have to do so before the next **FooDataReader_take** (p. 626) call. However, failure to return the loan will eventually deplete the **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615) is specified by the **DDS_ResourceLimitsQosPolicy** (p. 1949) and the **DDS_DataReaderResourceLimitsQosPolicy** (p. 1619).

If the sequences are not empty (maximum size not equal to 0 and length not equal to 0) when **FooDataReader_take** (p. 626) is called, samples are copied to `received_data` and `info_seq`. The application will not need to call **FooDataReader_return_loan** (p. 646).

The order of the samples returned to the caller depends on the **DDS_PresentationQosPolicy** (p. 1890).

- If **DDS_PresentationQosPolicy::access_scope** (p. 1893) is **DDS_INSTANCE_PRESENTATION_QOS** (p. 1305), the returned collection is a list where samples belonging to the same data instance are consecutive.

- If **DDS_PresentationQosPolicy::access_scope** (p. 1893) is **DDS_TOPIC_PRESENTATION_QOS** (p. 1305) and **DDS_PresentationQosPolicy::ordered_access** (p. 1893) is set to **DDS_BOOLEAN_FALSE** (p. 1201), then the returned collection is a list where samples belonging to the same data instance are consecutive.
- If **DDS_PresentationQosPolicy::access_scope** (p. 1893) is **DDS_TOPIC_PRESENTATION_QOS** (p. 1305) and **DDS_PresentationQosPolicy::ordered_access** (p. 1893) is set to **DDS_BOOLEAN_TRUE** (p. 1201), 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. 1893) is **DDS_GROUP_PRESENTATION_QOS** (p. 1305) and **DDS_PresentationQosPolicy::ordered_access** (p. 1893) is set to **DDS_BOOLEAN_FALSE** (p. 1201), then the returned collection is a list where samples belonging to the same data instance are consecutive.
- If **DDS_PresentationQosPolicy::access_scope** (p. 1893) is **DDS_GROUP_PRESENTATION_QOS** (p. 1305) and **DDS_PresentationQosPolicy::ordered_access** (p. 1893) is set to **DDS_BOOLEAN_TRUE** (p. 1201), 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 **DDS_Subscriber_begin_access** (p. 587) and **DDS_Subscriber_end_access** (p. 588) operations; see the "Ordered Access" section in the "PRESENTATION QosPolicy" section of the *Core Libraries User's Manual*.

In all of the above cases, the relative order between the samples of one instance is consistent with the **DESTINATION_ORDER** (p. 1273) policy:

- If **DDS_DestinationOrderQosPolicy::kind** (p. 1688) is **DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS** (p. 1274), 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. 1688) is **DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 1274), 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 **DDS_DataReader** (p. 615) has no samples that meet the constraints, the function will fail with **DDS_RETCODE_NO_DATA** (p. 1222).

In addition to the collection of samples, the read and take operations also use a collection of **DDS_SampleInfo** (p. 1985) 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. 1524))
- the current-length (`len`, see **FooSeq_get_length** (p. 1515))

- the maximum length (`max_len`, see **FooSeq_get_maximum** (p. 1514))

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. 1222).
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 **DDS_DataReader** (p. 615). 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 **DDS_DataWriter** (p. 485) using **FooDataReader_return_loan** (p. 646).
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. 1222). 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. 1985) 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 than 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. 1222). 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 **DDS_DataReader** (p. 615), 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 **DDS_DataReader** (p. 615). If this is the case, the application will need to use **FooDataReader_return_loan** (p. 646) to return the loan once it is no longer using the `received_data` in the collection. When **FooDataReader_return_loan** (p. 646) 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. 646), 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. 2109) and **DDS_SampleInfo** (p. 1985) 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. 2109) values and the collection of **DDS_SampleInfo** (p. 1985) structures are of the same length and are in a one-to-one correspondence. Each **DDS_SampleInfo** (p. 1985) 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. 1985)

is **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 715) or **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 715), then the last sample for that instance in the collection (that is, the one whose **DDS_SampleInfo** (p. 1985) 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. 1949). The act of reading/taking a sample sets its `sample_state` to **DDS_READ_SAMPLE_STATE** (p. 711).

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. 713). 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. 2109)).

If the **DDS_DataReader** (p. 615) has no samples that meet the constraints, the operations fails with **DDS_RETCODE_NO_DATA** (p. 1222). For an example on how `take` can be used, please refer to the **Access received data via a reader** (p. 987) "receive example".

Parameters

<i>received_data</i>	<< <i>inout</i> >> (p. 1013) User data type-specific FooSeq (p. 2113) object where the received data samples will be returned. Must be a valid non-NULL FooSeq (p. 2113). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>info_seq</i>	<< <i>inout</i> >> (p. 1013) A DDS_SampleInfoSeq (p. 1996) object where the received sample info will be returned. Must be a valid non-NULL DDS_SampleInfoSeq (p. 1996). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>max_samples</i>	<< <i>in</i> >> (p. 1013) The maximum number of samples to be returned. If the special value DDS_LENGTH_UNLIMITED (p. 1326) is provided, as many samples will be returned as are available, up to the limits described above.
<i>sample_states</i>	<< <i>in</i> >> (p. 1013) Data samples matching one of these <code>sample_states</code> are returned.
<i>view_states</i>	<< <i>in</i> >> (p. 1013) Data samples matching one of these <code>view_state</code> are returned.
<i>instance_states</i>	<< <i>in</i> >> (p. 1013) Data samples matching one of these <code>instance_state</code> are returned.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222), **DDS_RETCODE_NO_DATA** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

FooDataReader_read (p. 625)

FooDataReader_read_w_condition (p. 630), **FooDataReader_take_w_condition** (p. 632)

DDS_LENGTH_UNLIMITED (p. 1326)

5.21.5.5 FooDataReader_read_w_condition()

```
DDS_ReturnCode_t FooDataReader_read_w_condition (
    FooDataReader * self,
    struct FooSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    DDS_ReadCondition * condition )
```

Accesses via **FooDataReader_read** (p. 625) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).

This operation is especially useful in combination with **DDS_QueryCondition** (p. 699) to filter data samples based on the content.

The specified **DDS_ReadCondition** (p. 694) must be attached to the **DDS_DataReader** (p. 615); otherwise the operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

In case the **DDS_ReadCondition** (p. 694) is a plain **DDS_ReadCondition** (p. 694) and not the specialized **DDS_↵_QueryCondition** (p. 699), the operation is equivalent to calling **FooDataReader_read** (p. 625) 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 **DDS_ReadCondition** (p. 694).

The samples are accessed with the same semantics as **FooDataReader_read** (p. 625).

If the **DDS_DataReader** (p. 615) has no samples that meet the constraints, the operation will fail with **DDS_RETCODE_↵_NO_DATA** (p. 1222).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>received_data</i>	<< <i>inout</i> >> (p. 1013) user data type-specific FooSeq (p. 2113) object where the received data samples will be returned. Must be a valid non-NULL FooSeq (p. 2113). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>info_seq</i>	<< <i>inout</i> >> (p. 1013) a DDS_SampleInfoSeq (p. 1996) object where the received sample info will be returned. Must be a valid non-NULL DDS_SampleInfoSeq (p. 1996). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>max_samples</i>	<< <i>in</i> >> (p. 1013) The maximum number of samples to be returned. If the special value DDS_LENGTH_UNLIMITED (p. 1326) is provided, as many samples will be returned as are available, up to the limits described in the documentation for FooDataReader_take() (p. 626).
<i>condition</i>	<< <i>in</i> >> (p. 1013) the DDS_ReadCondition (p. 694) to select samples of interest. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222), **DDS_↵_RETCODE_NO_DATA** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

FooDataReader_read (p. 625)

FooDataReader_take (p. 626), **FooDataReader_take_w_condition** (p. 632)

DDS_LENGTH_UNLIMITED (p. 1326)

5.21.5.6 FooDataReader_take_w_condition()

```
DDS_ReturnCode_t FooDataReader_take_w_condition (
    FooDataReader * self,
    FooSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    DDS_ReadCondition * condition )
```

Analogous to **FooDataReader_read_w_condition** (p. 630) except it accesses samples via the **FooDataReader_take** (p. 626) operation.

This operation is analogous to **FooDataReader_read_w_condition** (p. 630) except that it accesses samples via the **FooDataReader_take** (p. 626) operation.

The specified **DDS_ReadCondition** (p. 694) must be attached to the **DDS_DataReader** (p. 615); otherwise the operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

The samples are accessed with the same semantics as **FooDataReader_take** (p. 626).

This operation is especially useful in combination with **DDS_QueryCondition** (p. 699) to filter data samples based on the content.

If the **DDS_DataReader** (p. 615) has no samples that meet the constraints, the function will fail with **DDS_RETCODE_←_NO_DATA** (p. 1222).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>received_data</i>	<< <i>inout</i> >> (p. 1013) user data type-specific FooSeq (p. 2113) object where the received data samples will be returned. Must be a valid non-NULL FooSeq (p. 2113). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>info_seq</i>	<< <i>inout</i> >> (p. 1013) a DDS_SampleInfoSeq (p. 1996) object where the received sample info will be returned. Must be a valid non-NULL DDS_SampleInfoSeq (p. 1996). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>max_samples</i>	<< <i>in</i> >> (p. 1013) The maximum number of samples to be returned. If the special value DDS_LENGTH_UNLIMITED (p. 1326) is provided, as many samples will be returned as are available, up to the limits described in the documentation for FooDataReader_take() (p. 626).
<i>condition</i>	<< <i>in</i> >> (p. 1013) the DDS_ReadCondition (p. 694) to select samples of interest. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222), **DDS_RETCODE_NO_DATA** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

FooDataReader_read_w_condition (p. 630), **FooDataReader_read** (p. 625)

FooDataReader_take (p. 626)

DDS_LENGTH_UNLIMITED (p. 1326)

5.21.5.7 FooDataReader_read_next_sample()

```
DDS_ReturnCode_t FooDataReader_read_next_sample (
    FooDataReader * self,
    struct Foo * received_data,
    struct DDS_SampleInfo * sample_info )
```

Copies the next not-previously-accessed data value from the **DDS_DataReader** (p. 615).

This operation copies the next not-previously-accessed data value from the **DDS_DataReader** (p. 615). This operation also copies the corresponding **DDS_SampleInfo** (p. 1985). The implied order among the samples stored in the **DDS_DataReader** (p. 615) is the same as for the **FooDataReader_read** (p. 625) operation.

The **FooDataReader_read_next_sample** (p. 633) operation is semantically equivalent to the **FooDataReader_read** (p. 625) 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. 633) 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 **DDS_DataReader** (p. 615), the operation will fail with **DDS_RETCODE_NO_DATA** (p. 1222) and nothing is copied.

Note

Calling **FooDataReader_read_next_sample** (p. 633) from the **DDS_DataReaderListener::on_data_available()** (p. 1595) callback reads only one sample of potentially many samples in the reader queue, because **DDS_DataReaderListener::on_data_available()** (p. 1595) is triggered only once when new samples arrive in the queue. Therefore, it is recommended that you call **FooDataReader_read_next_sample** (p. 633) in a loop within the **on_data_available** callback until **FooDataReader_read_next_sample** (p. 633) returns **DDS_RETCODE_NO_DATA** (p. 1222) . This ensures that all samples in the reader queue are serviced by application logic. (You may also choose to use **FooDataReader_read** (p. 625) rather than **FooDataReader_read_next_sample** (p. 633) in order to read more than one sample at a time.)

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>received_data</i> Generated by Doxygen	<< <i>inout</i> >> (p. 1013) user data type-specific Foo (p. 2109) object where the next received data sample will be returned. The received_data must have been fully allocated. Otherwise, this operation may fail. Must be a valid non-NULL Foo (p. 2109). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>sample_info</i>	<< <i>inout</i> >> (p. 1013) a DDS_SampleInfo (p. 1985) object where the next received sample info

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_NO_DATA** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

FooDataReader_read (p. 625)

5.21.5.8 FooDataReader_take_next_sample()

```
DDS_ReturnCode_t FooDataReader_take_next_sample (
    FooDataReader * self,
    struct Foo * received_data,
    struct DDS_SampleInfo * sample_info )
```

Copies the next not-previously-accessed data value from the **DDS_DataReader** (p. 615).

This operation copies the next not-previously-accessed data value from the **DDS_DataReader** (p. 615) and 'removes' it from the **DDS_DataReader** (p. 615) so that it is no longer accessible. This operation also copies the corresponding **DDS_SampleInfo** (p. 1985). This operation is analogous to the **FooDataReader_read_next_sample** (p. 633) except for the fact that the sample is removed from the **DDS_DataReader** (p. 615).

The **FooDataReader_take_next_sample** (p. 634) operation is semantically equivalent to the **FooDataReader_take** (p. 626) 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. 634) 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 **DDS_DataReader** (p. 615), the operation will fail with **DDS_RETCODE_NO_DATA** (p. 1222) and nothing is copied.

Note

Calling **FooDataReader_take_next_sample** (p. 634) from the **DDS_DataReaderListener::on_data_available()** (p. 1595) callback retrieves only one sample of potentially many samples in the reader queue, because **DDS_DataReaderListener::on_data_available()** (p. 1595) is triggered only once when new samples arrive in the queue. Therefore, it is recommended that you call **FooDataReader_take_next_sample** (p. 634) in a loop within the on_data_available callback until **FooDataReader_take_next_sample** (p. 634) returns **DDS_RETCODE_NO_DATA** (p. 1222). This ensures that all samples in the reader queue are serviced by application logic. (You may also choose to use the **FooDataReader_take** (p. 626) rather than **FooDataReader_take_next_sample** (p. 634) in order to take more than one sample at a time.)

Parameters

<i>self</i>	<<in>> (p. 1013) Cannot be NULL.
<i>received_data</i>	<<inout>> (p. 1013) user data type-specific Foo (p. 2109) object where the next received data sample will be returned. The received_data must have been fully allocated. Otherwise, this operation may fail. Must be a valid non-NULL Foo (p. 2109). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>sample_info</i>	<<inout>> (p. 1013) a DDS_SampleInfo (p. 1985) object where the next received sample info will be returned. Must be a valid non-NULL DDS_SampleInfo (p. 1985). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.

Returns

One of the **Standard Return Codes** (p. 1221),

DDS_RETCODE_NO_DATA (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

FooDataReader_take (p. 626)

5.21.5.9 FooDataReader_read_instance()

```
DDS_ReturnCode_t FooDataReader_read_instance (
    FooDataReader * self,
    struct FooSeq * received_data,
    struct 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 )
```

Access a collection of data samples from the **DDS_DataReader** (p. 615).

This operation accesses a collection of data values from the **DDS_DataReader** (p. 615). The behavior is identical to **FooDataReader_read** (p. 625), 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. 1985) verifies **DDS_SampleInfo::instance_handle** (p. 1990) == `a_handle`.

The **FooDataReader_read_instance** (p. 635) operation is semantically equivalent to the **FooDataReader_read** (p. 625) operation, except in building the collection, the **DDS_DataReader** (p. 615) 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. 635) operation follows the same rules as the **FooDataReader_read** (p. 625) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **FooDataReader_read** (p. 625), the **FooDataReader_read_instance** (p. 635) operation may 'loan' elements to the output collections, which must then be returned by means of **FooDataReader_return_loan** (p. 646).

Similar to the **FooDataReader_read** (p. 625), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **DDS_DataReader** (p. 615) has no samples that meet the constraints, the function will fail with **DDS_RETCODE_NO_DATA** (p. 1222).

This operation may fail with **DDS_RETCODE_BAD_PARAMETER** (p. 1222) if the **DDS_InstanceHandle_t** (p. 214) `a_handle` does not correspond to an existing data-object known to the **DDS_DataReader** (p. 615).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>received_data</i>	<< <i>inout</i> >> (p. 1013) user data type-specific FooSeq (p. 2113) object where the received data samples will be returned. Must be a valid non-NULL FooSeq (p. 2113). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>info_seq</i>	<< <i>inout</i> >> (p. 1013) a DDS_SampleInfoSeq (p. 1996) object where the received sample info will be returned. Must be a valid non-NULL DDS_SampleInfoSeq (p. 1996). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>max_samples</i>	<< <i>in</i> >> (p. 1013) The maximum number of samples to be returned. If the special value DDS_LENGTH_UNLIMITED (p. 1326) is provided, as many samples will be returned as are available, up to the limits described in the documentation for FooDataReader_take() (p. 626).
<i>a_handle</i>	<< <i>in</i> >> (p. 1013) The specified instance to return samples for. Must be a valid non-NULL DDS_InstanceHandle_t (p. 214). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL. The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if the <i>handle</i> does not correspond to an existing data-object known to the DDS_DataReader (p. 615).
<i>sample_states</i>	<< <i>in</i> >> (p. 1013) data samples matching ones of these <i>sample_states</i> are returned.
<i>view_states</i>	<< <i>in</i> >> (p. 1013) data samples matching ones of these <i>view_state</i> are returned.
<i>instance_states</i>	<< <i>in</i> >> (p. 1013) data samples matching ones of these <i>instance_state</i> are returned.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222), **DDS_RETCODE_NO_DATA** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

FooDataReader_read (p. 625)

DDS_LENGTH_UNLIMITED (p. 1326)

5.21.5.10 FooDataReader_take_instance()

```

DDS_ReturnCode_t FooDataReader_take_instance (
    FooDataReader * self,
    struct FooSeq * received_data,
    struct 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 )

```

Access a collection of data samples from the **DDS_DataReader** (p. 615).

This operation accesses a collection of data values from the **DDS_DataReader** (p. 615). The behavior is identical to **FooDataReader_take** (p. 626), 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. 626) operation, except in building the collection, the **DDS_DataReader** (p. 615) 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. 636) operation follows the same rules as the **FooDataReader_read** (p. 625) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **FooDataReader_read** (p. 625), the **FooDataReader_take_instance** (p. 636) operation may 'loan' elements to the output collections, which must then be returned by means of **FooDataReader_return_loan** (p. 646).

Similar to the **FooDataReader_read** (p. 625), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **DDS_DataReader** (p. 615) has no samples that meet the constraints, the function fails with **DDS_RETCODE_NO_DATA** (p. 1222).

This operation may fail with **DDS_RETCODE_BAD_PARAMETER** (p. 1222) if the **DDS_InstanceHandle_t** (p. 214) `a_handle` does not correspond to an existing data-object known to the **DDS_DataReader** (p. 615).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>received_data</i>	<< <i>inout</i> >> (p. 1013) user data type-specific FooSeq (p. 2113) object where the received data samples will be returned. Must be a valid non-NULL FooSeq (p. 2113). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>info_seq</i>	<< <i>inout</i> >> (p. 1013) a DDS_SampleInfoSeq (p. 1996) object where the received sample info will be returned. Must be a valid non-NULL DDS_SampleInfoSeq (p. 1996). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>max_samples</i>	<< <i>in</i> >> (p. 1013) The maximum number of samples to be returned. If the special value DDS_LENGTH_UNLIMITED (p. 1326) is provided, as many samples will be returned as are available, up to the limits described in the documentation for FooDataReader_take() (p. 626).
<i>a_handle</i>	<< <i>in</i> >> (p. 1013) The specified instance to return samples for. Must be a valid non-NULL DDS_InstanceHandle_t (p. 214). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL. The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if the <code>handle</code> does not correspond to an existing data-object known to the DDS_DataReader (p. 615).
<i>sample_states</i>	<< <i>in</i> >> (p. 1013) data samples matching ones of these <code>sample_states</code> are returned.
<i>view_states</i>	<< <i>in</i> >> (p. 1013) data samples matching ones of these <code>view_state</code> are returned.
<i>instance_states</i>	<< <i>in</i> >> (p. 1013) data samples matching ones of these <code>instance_state</code> are returned.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222), **DDS_RETCODE_NO_DATA** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

FooDataReader_take (p. 626)

DDS_LENGTH_UNLIMITED (p. 1326)

5.21.5.11 FooDataReader_read_instance_w_condition()

```
DDS_ReturnCode_t FooDataReader_read_instance_w_condition (
    FooDataReader * self,
    struct FooSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    const DDS_InstanceHandle_t * a_handle,
    DDS_ReadCondition * condition )
```

<<**extension**>> (p. 1012) Accesses via **FooDataReader_read_instance** (p. 635) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).

This operation accesses a collection of data values from the **DDS_DataReader** (p. 615). The behavior is identical to **FooDataReader_read_instance** (p. 635), 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 **DDS_ReadCondition** (p. 694) evaluates to TRUE.

The behavior of the **FooDataReader_read_instance_w_condition** (p. 638) operation follows the same rules as the **FooDataReader_read** (p. 625) operation regarding the pre-conditions and post-conditions for the *received_data* and *sample_info*. Similar to the **FooDataReader_read** (p. 625), the **FooDataReader_read_instance_w_condition** (p. 638) operation may 'loan' elements to the output collections, which must then be returned by means of **FooDataReader_return_loan** (p. 646).

Similar to **FooDataReader_read** (p. 625), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **DDS_DataReader** (p. 615) has no samples that meet the constraints, the function will fail with **DDS_RETCODE_NO_DATA** (p. 1222).

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>received_data</i>	<< inout >> (p. 1013) user data type-specific FooSeq (p. 2113) object where the received data samples will be returned. Must be a valid non-NULL FooSeq (p. 2113). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>info_seq</i>	<< inout >> (p. 1013) a DDS_SampleInfoSeq (p. 1996) object where the received sample info will be returned. Must be a valid non-NULL DDS_SampleInfoSeq (p. 1996). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>max_samples</i>	<< in >> (p. 1013) The maximum number of samples to be returned. If the special value DDS_LENGTH_UNLIMITED (p. 1326) is provided, as many samples will be returned as are available, up to the limits described in the documentation for FooDataReader_take() (p. 626).
<i>a_handle</i>	<< in >> (p. 1013) The 'next smallest' instance with a value greater than this value that has available samples will be returned. Must be a valid non-NULL DDS_InstanceHandle_t (p. 214). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>condition</i>	<< in >> (p. 1013) the DDS_ReadCondition (p. 694) to select samples of interest. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) **DDS_←
RETCODE_NO_DATA** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

FooDataReader_read_next_instance (p. 640)

DDS_LENGTH_UNLIMITED (p. 1326)

5.21.5.12 FooDataReader_take_instance_w_condition()

```
DDS_ReturnCode_t FooDataReader_take_instance_w_condition (
    FooDataReader * self,
    struct FooSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    const DDS_InstanceHandle_t * a_handle,
    DDS_ReadCondition * condition )
```

<<**extension**>> (p. 1012) Accesses via **FooDataReader_take_instance** (p. 636) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).

This operation accesses a collection of data values from the **DDS_DataReader** (p. 615) and 'removes' them from the **DDS_DataReader** (p. 615). The behavior is identical to **FooDataReader_take_instance** (p. 636), 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 **DDS_ReadCondition** (p. 694) evaluates to TRUE.

The operation has the same behavior as **FooDataReader_read_instance_w_condition** (p. 638), except that the samples are 'taken' from the **DDS_DataReader** (p. 615) such that they are no longer accessible via subsequent 'read' or 'take' operations.

The behavior of the **FooDataReader_take_instance_w_condition** (p. 639) operation follows the same rules as the **FooDataReader_read** (p. 625) operation regarding the pre-conditions and post-conditions for the *received_data* and *sample_info*. Similar to the **FooDataReader_read** (p. 625), the **FooDataReader_take_instance_w_condition** (p. 639) operation may 'loan' elements to the output collections, which must then be returned by means of **FooData←
Reader_return_loan** (p. 646).

Similar to the **FooDataReader_read** (p. 625), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **DDS_DataReader** (p. 615) has no samples that meet the constraints, the function will fail with **DDS_RETCODE_←
_NO_DATA** (p. 1222).

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.

Parameters

<i>received_data</i>	<< <i>inout</i> >> (p. 1013) user data type-specific FooSeq (p. 2113) object where the received data samples will be returned. Must be a valid non-NULL FooSeq (p. 2113). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>info_seq</i>	<< <i>inout</i> >> (p. 1013) a DDS_SampleInfoSeq (p. 1996) object where the received sample info will be returned. Must be a valid non-NULL DDS_SampleInfoSeq (p. 1996). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>max_samples</i>	<< <i>in</i> >> (p. 1013) The maximum number of samples to be returned. If the special value DDS_LENGTH_UNLIMITED (p. 1326) is provided, as many samples will be returned as are available, up to the limits described in the documentation for FooDataReader_take() (p. 626).
<i>a_handle</i>	<< <i>in</i> >> (p. 1013) The 'next smallest' instance with a value greater than this value that has available samples will be returned. Must be a valid non-NULL DDS_InstanceHandle_t (p. 214). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>condition</i>	<< <i>in</i> >> (p. 1013) the DDS_ReadCondition (p. 694) to select samples of interest. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222), or **DDS_RETCODE_NO_DATA** (p. 1222), **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

FooDataReader_take_next_instance (p. 642)

DDS_LENGTH_UNLIMITED (p. 1326)

5.21.5.13 FooDataReader_read_next_instance()

```
DDS_ReturnCode_t FooDataReader_read_next_instance (
    FooDataReader * self,
    struct FooSeq * received_data,
    struct 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 )
```

Access a collection of data samples from the **DDS_DataReader** (p. 615).

This operation accesses a collection of data values from the **DDS_DataReader** (p. 615) where all the samples belong to a single instance. The behavior is similar to **FooDataReader_read_instance** (p. 635), 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 **DDS_DataReader** (p. 615). 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. 640) is 'as if' the **DDS_DataReader** (p. 615) invoked **FooDataReader_read_instance** (p. 635), 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. 228) is guaranteed to be 'less than' any valid `instance_handle`. So the use of the parameter value `previous_handle == DDS_HANDLE_NIL` (p. 228) 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. 640) is intended to be used in an application-driven iteration, where the application starts by passing `previous_handle == DDS_HANDLE_NIL` (p. 228), examines the samples returned, and then uses the `instance_handle` returned in the **DDS_SampleInfo** (p. 1985) as the value of the `previous_handle` argument to the next call to **FooDataReader_read_next_instance** (p. 640). The iteration continues until **FooDataReader_read_next_instance** (p. 640) fails with **DDS_RETCODE_NO_DATA** (p. 1222). 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. 640) operation with a `previous_handle` that does not correspond to an instance currently managed by the **DDS_DataReader** (p. 615). This is because as stated earlier the 'greater-than' relationship is defined even for handles not managed by the **DDS_DataReader** (p. 615). 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. 715) 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. 640) operation follows the same rules as the **FooDataReader_read** (p. 625) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **FooDataReader_read** (p. 625), the **FooDataReader_read_instance** (p. 635) operation may 'loan' elements to the output collections, which must then be returned by means of **FooDataReader_return_loan** (p. 646).

Similar to the **FooDataReader_read** (p. 625), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **DDS_DataReader** (p. 615) has no samples that meet the constraints, the function will fail with **DDS_RETCODE_NO_DATA** (p. 1222).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>received_data</i>	<< <i>inout</i> >> (p. 1013) user data type-specific FooSeq (p. 2113) object where the received data samples will be returned. Must be a valid non-NULL FooSeq (p. 2113). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>info_seq</i>	<< <i>inout</i> >> (p. 1013) a DDS_SampleInfoSeq (p. 1996) object where the received sample info will be returned. Must be a valid non-NULL DDS_SampleInfoSeq (p. 1996). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.

Parameters

<i>max_samples</i>	<< <i>in</i> >> (p. 1013) The maximum number of samples to be returned. If the special value DDS_LENGTH_UNLIMITED (p. 1326) is provided, as many samples will be returned as are available, up to the limits described in the documentation for FooDataReader_take() (p. 626).
<i>previous_handle</i>	<< <i>in</i> >> (p. 1013) The 'next smallest' instance with a value greater than this value that has available samples will be returned. Must be a valid non-NULL DDS_InstanceHandle_t (p. 214). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>sample_states</i>	<< <i>in</i> >> (p. 1013) data samples matching ones of these <i>sample_states</i> are returned. See the valid values for this parameter here: DDS_SampleStateKind (p. 710).
<i>view_states</i>	<< <i>in</i> >> (p. 1013) data samples matching ones of these <i>view_state</i> are returned. See the valid values for this parameter here: DDS_ViewStateKind (p. 712).
<i>instance_states</i>	<< <i>in</i> >> (p. 1013) data samples matching ones of these <i>instance_state</i> are returned. See the valid values for this parameter here: DDS_InstanceStateKind (p. 714).

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) **DDS_↵
RETCODE_NO_DATA** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

FooDataReader_read (p. 625)

DDS_LENGTH_UNLIMITED (p. 1326)

5.21.5.14 FooDataReader_take_next_instance()

```
DDS_ReturnCode_t FooDataReader_take_next_instance (
    FooDataReader * self,
    struct FooSeq * received_data,
    struct 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 )
```

Access a collection of data samples from the **DDS_DataReader** (p. 615).

This operation accesses a collection of data values from the **DDS_DataReader** (p. 615) and 'removes' them from the **DDS_DataReader** (p. 615).

This operation has the same behavior as **FooDataReader_read_next_instance** (p. 640), except that the samples are 'taken' from the **DDS_DataReader** (p. 615) such that they are no longer accessible via subsequent 'read' or 'take' operations.

Note

Like **FooDataReader_read_next_instance** (p. 640), 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. 642) fails with the value **DDS_RETCODE_NO_DATA** (p. 1222) .

Similar to the operation **FooDataReader_read_next_instance** (p. 640), it is possible to call **FooDataReader_take_next_instance** (p. 642) with a `previous_handle` that does not correspond to an instance currently managed by the **DDS_DataReader** (p. 615).

The behavior of the **FooDataReader_take_next_instance** (p. 642) operation follows the same rules as the **FooDataReader_read** (p. 625) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **FooDataReader_read** (p. 625), the **FooDataReader_take_next_instance** (p. 642) operation may 'loan' elements to the output collections, which must then be returned by means of **FooDataReader_return_loan** (p. 646).

Similar to the **FooDataReader_read** (p. 625), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **DDS_DataReader** (p. 615) has no samples that meet the constraints, the function will fail with **DDS_RETCODE_NO_DATA** (p. 1222).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>received_data</i>	<< <i>inout</i> >> (p. 1013) user data type-specific FooSeq (p. 2113) object where the received data samples will be returned. Must be a valid non-NULL FooSeq (p. 2113). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>info_seq</i>	<< <i>inout</i> >> (p. 1013) a DDS_SampleInfoSeq (p. 1996) object where the received sample info will be returned. Must be a valid non-NULL DDS_SampleInfoSeq (p. 1996). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>max_samples</i>	<< <i>in</i> >> (p. 1013) The maximum number of samples to be returned. If the special value DDS_LENGTH_UNLIMITED (p. 1326) is provided, as many samples will be returned as are available, up to the limits described in the documentation for FooDataReader_take() (p. 626).
<i>previous_handle</i>	<< <i>in</i> >> (p. 1013) The 'next smallest' instance with a value greater than this value that has available samples will be returned. Must be a valid non-NULL DDS_InstanceHandle_t (p. 214). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>sample_states</i>	<< <i>in</i> >> (p. 1013) data samples matching ones of these <code>sample_states</code> are returned.
<i>view_states</i>	<< <i>in</i> >> (p. 1013) data samples matching ones of these <code>view_state</code> are returned.
<i>instance_states</i>	<< <i>in</i> >> (p. 1013) data samples matching ones of these <code>instance_state</code> are returned.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222), **DDS_RETCODE_NO_DATA** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

FooDataReader_take (p. 626)

DDS_LENGTH_UNLIMITED (p. 1326)

5.21.5.15 FooDataReader_read_next_instance_w_condition()

```
DDS_ReturnCode_t FooDataReader_read_next_instance_w_condition (
    FooDataReader * self,
    struct FooSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    const DDS_InstanceHandle_t * previous_handle,
    DDS_ReadCondition * condition )
```

Accesses via **FooDataReader_read_next_instance** (p. 640) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).

This operation accesses a collection of data values from the **DDS_DataReader** (p. 615). The behavior is identical to **FooDataReader_read_next_instance** (p. 640), 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 **DDS_ReadCondition** (p. 694) evaluates to TRUE.

Similar to the operation **FooDataReader_read_next_instance** (p. 640), it is possible to call **FooDataReader_read_next_instance_w_condition** (p. 643) with a *previous_handle* that does not correspond to an instance currently managed by the **DDS_DataReader** (p. 615).

The behavior of the **FooDataReader_read_next_instance_w_condition** (p. 643) operation follows the same rules as the **FooDataReader_read** (p. 625) operation regarding the pre-conditions and post-conditions for the *received_data* and *sample_info*. Similar to the **FooDataReader_read** (p. 625), the **FooDataReader_read_next_instance_w_condition** (p. 643) operation may 'loan' elements to the output collections, which must then be returned by means of **FooDataReader_return_loan** (p. 646).

Similar to the **FooDataReader_read** (p. 625), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **DDS_DataReader** (p. 615) has no samples that meet the constraints, the function will fail with **DDS_RETCODE_NO_DATA** (p. 1222).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>received_data</i>	<< <i>inout</i> >> (p. 1013) user data type-specific FooSeq (p. 2113) object where the received data samples will be returned. Must be a valid non-NULL FooSeq (p. 2113). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>info_seq</i>	<< <i>inout</i> >> (p. 1013) a DDS_SampleInfoSeq (p. 1996) object where the received sample info will be returned. Must be a valid non-NULL DDS_SampleInfoSeq (p. 1996). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>max_samples</i>	<< <i>in</i> >> (p. 1013) The maximum number of samples to be returned. If the special value DDS_LENGTH_UNLIMITED (p. 1326) is provided, as many samples will be returned as are available, up to the limits described in the documentation for FooDataReader_take() (p. 626).
<i>previous_handle</i>	<< <i>in</i> >> (p. 1013) The 'next smallest' instance with a value greater than this value that has available samples will be returned. Must be a valid non-NULL DDS_InstanceHandle_t (p. 214). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>condition</i>	<< <i>in</i> >> (p. 1013) the DDS_ReadCondition (p. 694) to select samples of interest. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) **DDS_↵
RETCODE_NO_DATA** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

FooDataReader_read_next_instance (p. 640)

DDS_LENGTH_UNLIMITED (p. 1326)

5.21.5.16 FooDataReader_take_next_instance_w_condition()

```
DDS_ReturnCode_t FooDataReader_take_next_instance_w_condition (
    FooDataReader * self,
    struct FooSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    const DDS_InstanceHandle_t * previous_handle,
    DDS_ReadCondition * condition )
```

Accesses via **FooDataReader_take_next_instance** (p. 642) the samples that match the criteria specified in the **DDS_↵
_ReadCondition** (p. 694).

This operation accesses a collection of data values from the **DDS_DataReader** (p. 615) and 'removes' them from the **DDS_DataReader** (p. 615).

The operation has the same behavior as **FooDataReader_read_next_instance_w_condition** (p. 643), except that the samples are 'taken' from the **DDS_DataReader** (p. 615) such that they are no longer accessible via subsequent 'read' or 'take' operations.

Similar to the operation **FooDataReader_read_next_instance** (p. 640), it is possible to call **FooDataReader_take_↵
next_instance_w_condition** (p. 645) with a `previous_handle` that does not correspond to an instance currently managed by the **DDS_DataReader** (p. 615).

The behavior of the **FooDataReader_take_next_instance_w_condition** (p. 645) operation follows the same rules as the **FooDataReader_read** (p. 625) operation regarding the pre-conditions and post-conditions for the `received_↵
data` and `sample_info`. Similar to **FooDataReader_read** (p. 625), the **FooDataReader_take_next_instance_↵
w_condition** (p. 645) operation may 'loan' elements to the output collections, which must then be returned by means of **FooDataReader_return_loan** (p. 646).

Similar to the **FooDataReader_read** (p. 625), this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

If the **DDS_DataReader** (p. 615) has no samples that meet the constraints, the function will fail with **DDS_RETCODE_↵
_NO_DATA** (p. 1222).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Parameters

<i>received_data</i>	<< <i>inout</i> >> (p. 1013) user data type-specific FooSeq (p. 2113) object where the received data samples will be returned. Must be a valid non-NULL FooSeq (p. 2113). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>info_seq</i>	<< <i>inout</i> >> (p. 1013) a DDS_SampleInfoSeq (p. 1996) object where the received sample info will be returned. Must be a valid non-NULL DDS_SampleInfoSeq (p. 1996). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>max_samples</i>	<< <i>in</i> >> (p. 1013) The maximum number of samples to be returned. If the special value DDS_LENGTH_UNLIMITED (p. 1326) is provided, as many samples will be returned as are available, up to the limits described in the documentation for FooDataReader_take() (p. 626).
<i>previous_handle</i>	<< <i>in</i> >> (p. 1013) The 'next smallest' instance with a value greater than this value that has available samples will be returned. Must be a valid non-NULL DDS_InstanceHandle_t (p. 214). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>condition</i>	<< <i>in</i> >> (p. 1013) the DDS_ReadCondition (p. 694) to select samples of interest. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222), or **DDS_RETCODE_NO_DATA** (p. 1222), **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

FooDataReader_take_next_instance (p. 642)

DDS_LENGTH_UNLIMITED (p. 1326)

5.21.5.17 FooDataReader_return_loan()

```
DDS_ReturnCode_t FooDataReader_return_loan (
    FooDataReader * self,
    FooSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq )
```

Indicates to the **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615).

This operation indicates to the **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615).

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 **DDS_DataReader** (p. 615) to which they are returned. If either of these conditions is not met, the operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

The operation **FooDataReader_return_loan** (p. 646) allows implementations of the read and take operations to "loan" buffers from the **DDS_DataReader** (p. 615) to the application and in this manner provide "zerocopy" access to the data. During the loan, the **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615), the application should not retain them indefinitely.

The use of **FooDataReader_return_loan** (p. 646) 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. 646) 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. 646), 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

<i>received_data</i>	<< <i>in</i> >> (p. 1013) user data type-specific FooSeq (p. 2113) object where the received data samples was obtained from earlier invocation of read or take on the DDS_DataReader (p. 615). Must be a valid non-NULL FooSeq (p. 2113). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>info_seq</i>	<< <i>in</i> >> (p. 1013) a DDS_SampleInfoSeq (p. 1996) object where the received sample info was obtained from earlier invocation of read or take on the DDS_DataReader (p. 615). Must be a valid non-NULL DDS_SampleInfoSeq (p. 1996). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

5.21.5.18 FooDataReader_get_key_value()

```
DDS_ReturnCode_t FooDataReader_get_key_value (
    FooDataReader * self,
    Foo * key_holder,
    const DDS_InstanceHandle_t * handle )
```

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 function has no effect and exits with no error.

For keyed data types, this operation may fail with **DDS_RETCODE_BAD_PARAMETER** (p. 1222) if the `handle` does not correspond to an existing data-object known to the **DDS_DataReader** (p. 615).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>key_holder</i>	<< <i>inout</i> >> (p. 1013) a user data type specific key holder, whose <code>key</code> fields are filled by this operation. If Foo (p. 2109) has no key, this function has no effect. This function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if <code>key_holder</code> is NULL.
<i>handle</i>	<< <i>in</i> >> (p. 1013) the <code>instance</code> whose key is to be retrieved. If Foo (p. 2109) has a key, <code>handle</code> must represent an existing instance of type Foo (p. 2109) known to the DDS_DataReader (p. 615). Otherwise, this function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222).

If **Foo** (p. 2109) has a key and `handle` is **DDS_HANDLE_NIL** (p. 228), this function will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 1222).

If **Foo** (p. 2109) has a key and `handle` represents an instance of another type or an instance of type **Foo** (p. 2109) that has been unregistered, this function will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 1222). If **Foo** (p. 2109) has no key, this function has no effect.

This function will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 1222) if `handle` is NULL.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

See also

FooDataWriter_get_key_value (p. 505)

5.21.5.19 FooDataReader_lookup_instance()

```
DDS_InstanceHandle_t FooDataReader_lookup_instance (
    FooDataReader * self,
    const Foo * key_holder )
```

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. 228).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>key_holder</i>	<< <i>in</i> >> (p. 1013) a user data type specific key holder.

Returns

the instance handle associated with this instance. If **Foo** (p.2109) has no key, this function has no effect and returns **DDS_HANDLE_NIL** (p.228)

5.21.5.20 FooDataReader_is_data_consistent()

```
DDS_ReturnCode_t FooDataReader_is_data_consistent (
    FooDataReader * self,
    DDS_Boolean * is_data_consistent,
    const Foo * sample,
    const struct DDS_SampleInfo * sample_info )
```

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.210), 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. 649) 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.1682) 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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>is_data_consistent</i>	<< <i>inout</i> >> (p. 1013) Set to true if the sample is consistent (i.e., the sample has not been overwritten by the DataWriter)
<i>sample</i>	<< <i>in</i> >> (p. 1013) Sample to be validated
<i>sample_info</i>	<< <i>in</i> >> (p. 1013) DDS_SampleInfo (p. 1985) object received with the sample

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

5.21.5.21 DDS_RequestedDeadlineMissedStatus_initialize()

```
DDS_ReturnCode_t DDS_RequestedDeadlineMissedStatus_initialize (
    struct DDS_RequestedDeadlineMissedStatus * self )
```

Initializer for new status instances.

New **DDS_RequestedDeadlineMissedStatus** (p. 1946) instances on heap should be initialized with this function before they are passed to any function. This step ensures that those contained fields that use dynamic memory are properly initialized. This function does not allocate memory.

DDS_RequestedDeadlineMissedStatus_finalize (p. 651) should be called to free the contained fields that use dynamic memory:

```
DDS_RequestedDeadlineMissedStatus *myStatus = malloc(sizeof(struct DDS_RequestedDeadlineMissedStatus));
```

```
DDS_RequestedDeadlineMissedStatus_initialize(myStatus);
```

```
...
```

```
DDS_RequestedDeadlineMissedStatus_finalize(myStatus);
```

```
free(myStatus);
```

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

See also

DDS_RequestedDeadlineMissedStatus_INITIALIZER (p. 610)

DDS_RequestedDeadlineMissedStatus_finalize (p. 651)

5.21.5.22 DDS_RequestedDeadlineMissedStatus_copy()

```
DDS_ReturnCode_t DDS_RequestedDeadlineMissedStatus_copy (
    struct DDS_RequestedDeadlineMissedStatus * self,
    const struct DDS_RequestedDeadlineMissedStatus * source )
```

Copy the contents of the given status into this status.

Status instances can use dynamic memory because of the sequences contained in some status. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). Status to be copied from.

5.21.5.23 DDS_RequestedDeadlineMissedStatus_finalize()

```
DDS_ReturnCode_t DDS_RequestedDeadlineMissedStatus_finalize (
    struct DDS_RequestedDeadlineMissedStatus * self )
```

Free any dynamic memory allocated by status instances.

some status may use dynamic memory (regardless of whether the status itself is in dynamic memory). This function frees that memory but otherwise leaves this status unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

Note that if this status instance is stored in heap memory, calling this function will *not* call free() on it; the user is responsible for explicitly freeing any heap-based status instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.21.5.24 DDS_RequestedDeadlineMissedStatus_equals()

```
DDS_Boolean DDS_RequestedDeadlineMissedStatus_equals (
    const struct DDS_RequestedDeadlineMissedStatus * left,
    const struct DDS_RequestedDeadlineMissedStatus * right )
```

Compares two **DDS_RequestedDeadlineMissedStatus** (p. 1946) for equality.

Parameters

<i>left</i>	<< <i>in</i> >> (p. 1013) This RequestedDeadlineMissedStatus. Can be NULL.
<i>right</i>	<< <i>in</i> >> (p. 1013) The other RequestedDeadlineMissedStatus to be compared with this status instance. Can be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two RequestedDeadlineMissedStatus have equal values, or **DDS_↔
BOOLEAN_FALSE** (p. 1201) otherwise.

5.21.5.25 DDS_LivelinessChangedStatus_initialize()

```
DDS_ReturnCode_t DDS_LivelinessChangedStatus_initialize (
    struct DDS_LivelinessChangedStatus * self )
```

Initializer for new status instances.

New **DDS_LivelinessChangedStatus** (p. 1820) instances on heap should be initialized with this function before they are passed to any function. This step ensures that those contained fields that use dynamic memory are properly initialized. This function does not allocate memory.

DDS_LivelinessChangedStatus_finalize (p. 652) should be called to free the contained fields that use dynamic memory:

```
DDS_LivelinessChangedStatus *myStatus = malloc(sizeof(struct DDS_LivelinessChangedStatus));

DDS_LivelinessChangedStatus_initialize(myStatus);

...

DDS_LivelinessChangedStatus_finalize(myStatus);

free(myStatus);
```

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

See also

DDS_LivelinessChangedStatus_INITIALIZER (p. 611)

DDS_LivelinessChangedStatus_finalize (p. 652)

5.21.5.26 DDS_LivelinessChangedStatus_copy()

```
DDS_ReturnCode_t DDS_LivelinessChangedStatus_copy (
    struct DDS_LivelinessChangedStatus * self,
    const struct DDS_LivelinessChangedStatus * source )
```

Copy the contents of the given status into this status.

Status instances can use dynamic memory because of the sequences contained in some status. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). Status to be copied from.

5.21.5.27 DDS_LivelinessChangedStatus_finalize()

```
DDS_ReturnCode_t DDS_LivelinessChangedStatus_finalize (
    struct DDS_LivelinessChangedStatus * self )
```

Free any dynamic memory allocated by status instances.

some status may use dynamic memory (regardless of whether the status itself is in dynamic memory). This function frees that memory but otherwise leaves this status unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

Note that if this status instance is stored in heap memory, calling this function will *not* call free() on it; the user is responsible for explicitly freeing any heap-based status instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.21.5.28 DDS_LivelinessChangedStatus_equals()

```
DDS_Boolean DDS_LivelinessChangedStatus_equals (
    const struct DDS_LivelinessChangedStatus * left,
    const struct DDS_LivelinessChangedStatus * right )
```

Compares two **DDS_LivelinessChangedStatus** (p. 1820) for equality.

Parameters

<i>left</i>	<< <i>in</i> >> (p. 1013) This LivelinessChangedStatus. Can be NULL.
<i>right</i>	<< <i>in</i> >> (p. 1013) The other LivelinessChangedStatus to be compared with this status instance. Can be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two LivelinessChangedStatus have equal values, or **DDS_BOOLEAN_↵**
FALSE (p. 1201) otherwise.

5.21.5.29 DDS_RequestedIncompatibleQosStatus_initialize()

```
DDS_ReturnCode_t DDS_RequestedIncompatibleQosStatus_initialize (
    struct DDS_RequestedIncompatibleQosStatus * self )
```

Initializer for new status instances.

New **DDS_RequestedIncompatibleQosStatus** (p. 1947) instances on heap should be initialized with this function before they are passed to any function. This step ensures that those contained fields that use dynamic memory are properly initialized. This function does not allocate memory.

DDS_RequestedIncompatibleQosStatus_finalize (p. 654) should be called to free the contained fields that use dynamic memory:

```
DDS_RequestedIncompatibleQosStatus *myStatus = malloc(sizeof(struct DDS_RequestedIncompatibleQosStatus));

DDS_RequestedIncompatibleQosStatus_initialize(myStatus);

...

DDS_RequestedIncompatibleQosStatus_finalize(myStatus);

free(myStatus);
```

See also

DDS_RequestedIncompatibleQosStatus_INITIALIZER (p. 611)

DDS_RequestedIncompatibleQosStatus_finalize (p. 654)

5.21.5.30 DDS_RequestedIncompatibleQosStatus_copy()

```
DDS_ReturnCode_t DDS_RequestedIncompatibleQosStatus_copy (
    struct DDS_RequestedIncompatibleQosStatus * self,
    const struct DDS_RequestedIncompatibleQosStatus * source )
```

Copy the contents of the given status into this status.

Status instances can use dynamic memory because of the sequences contained in some status. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). Status to be copied from.

5.21.5.31 DDS_RequestedIncompatibleQosStatus_finalize()

```
DDS_ReturnCode_t DDS_RequestedIncompatibleQosStatus_finalize (
    struct DDS_RequestedIncompatibleQosStatus * self )
```

Free any dynamic memory allocated by status instances.

some status may use dynamic memory (regardless of whether the status itself is in dynamic memory). This function frees that memory but otherwise leaves this status unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

Note that if this status instance is stored in heap memory, calling this function will *not* call `free()` on it; the user is responsible for explicitly freeing any heap-based status instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.21.5.32 DDS_RequestedIncompatibleQosStatus_equals()

```
DDS_Boolean DDS_RequestedIncompatibleQosStatus_equals (
    const struct DDS_RequestedIncompatibleQosStatus * left,
    const struct DDS_RequestedIncompatibleQosStatus * right )
```

Compares two **DDS_RequestedIncompatibleQosStatus** (p. 1947) for equality.

Parameters

<i>left</i>	<< <i>in</i> >> (p. 1013) This RequestedIncompatibleQosStatus. Can be NULL.
<i>right</i>	<< <i>in</i> >> (p. 1013) The other RequestedIncompatibleQosStatus to be compared with this status instance. Can be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two RequestedIncompatibleQosStatus have equal values, or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.21.5.33 DDS_SampleLostStatus_initialize()

```
DDS_ReturnCode_t DDS_SampleLostStatus_initialize (
    struct DDS_SampleLostStatus * self )
```

Initializer for new status instances.

New **DDS_SampleLostStatus** (p. 1996) instances on heap should be initialized with this function before they are passed to any function. This step ensures that those contained fields that use dynamic memory are properly initialized. This function does not allocate memory.

DDS_RequestedIncompatibleQosStatus_finalize (p. 654) should be called to free the contained fields that use dynamic memory:

```

DDS_SampleLostStatus *myStatus = malloc(sizeof(struct DDS_SampleLostStatus));

DDS_SampleLostStatus_initialize(myStatus);

...

DDS_SampleLostStatus_finalize(myStatus);

free(myStatus);

```

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

See also

DDS_SampleLostStatus_INITIALIZER (p. 612)

DDS_SampleLostStatus_finalize (p. 656)

5.21.5.34 DDS_SampleLostStatus_copy()

```

DDS_ReturnCode_t DDS_SampleLostStatus_copy (
    struct DDS_SampleLostStatus * self,
    const struct DDS_SampleLostStatus * source )

```

Copy the contents of the given status into this status.

Status instances can use dynamic memory because of the sequences contained in some status. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). Status to be copied from.

5.21.5.35 DDS_SampleLostStatus_finalize()

```

DDS_ReturnCode_t DDS_SampleLostStatus_finalize (
    struct DDS_SampleLostStatus * self )

```

Free any dynamic memory allocated by status instances.

some status may use dynamic memory (regardless of whether the status itself is in dynamic memory). This function frees that memory but otherwise leaves this status unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

Note that if this status instance is stored in heap memory, calling this function will *not* call free() on it; the user is responsible for explicitly freeing any heap-based status instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.21.5.36 DDS_SampleLostStatus_equals()

```
DDS_Boolean DDS_SampleLostStatus_equals (
    const struct DDS_SampleLostStatus * left,
    const struct DDS_SampleLostStatus * right )
```

Compares two **DDS_SampleLostStatus** (p. 1996) for equality.

Parameters

<i>left</i>	<< <i>in</i> >> (p. 1013) This SampleLostStatus. Can be NULL.
<i>right</i>	<< <i>in</i> >> (p. 1013) The other SampleLostStatus to be compared with this status instance. Can be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two SampleLostStatus have equal values, or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.21.5.37 DDS_SampleRejectedStatus_initialize()

```
DDS_ReturnCode_t DDS_SampleRejectedStatus_initialize (
    struct DDS_SampleRejectedStatus * self )
```

Initializer for new status instances.

New **DDS_SampleRejectedStatus** (p. 1997) instances on heap should be initialized with this function before they are passed to any function. This step ensures that those contained fields that use dynamic memory are properly initialized. This function does not allocate memory.

DDS_RequestedIncompatibleQosStatus_finalize (p. 654) should be called to free the contained fields that use dynamic memory:

```
DDS_SampleRejectedStatus *myStatus = malloc(sizeof(struct DDS_SampleRejectedStatus));

DDS_SampleRejectedStatus_initialize(myStatus);

...

DDS_SampleRejectedStatus_finalize(myStatus);

free(myStatus);
```

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

See also

DDS_SampleRejectedStatus_INITIALIZER (p. 612)

DDS_SampleRejectedStatus_finalize (p. 658)

5.21.5.38 DDS_SampleRejectedStatus_copy()

```
DDS_ReturnCode_t DDS_SampleRejectedStatus_copy (
    struct DDS_SampleRejectedStatus * self,
    const struct DDS_SampleRejectedStatus * source )
```

Copy the contents of the given status into this status.

Status instances can use dynamic memory because of the sequences contained in some status. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). Status to be copied from.

5.21.5.39 DDS_SampleRejectedStatus_finalize()

```
DDS_ReturnCode_t DDS_SampleRejectedStatus_finalize (
    struct DDS_SampleRejectedStatus * self )
```

Free any dynamic memory allocated by status instances.

some status may use dynamic memory (regardless of whether the status itself is in dynamic memory). This function frees that memory but otherwise leaves this status unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

Note that if this status instance is stored in heap memory, calling this function will *not* call free() on it; the user is responsible for explicitly freeing any heap-based status instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.21.5.40 DDS_SampleRejectedStatus_equals()

```
DDS_Boolean DDS_SampleRejectedStatus_equals (
    const struct DDS_SampleRejectedStatus * left,
    const struct DDS_SampleRejectedStatus * right )
```

Compares two **DDS_SampleRejectedStatus** (p. 1997) for equality.

Parameters

<i>left</i>	<< <i>in</i> >> (p. 1013) This SampleRejectedStatus. Can be NULL.
<i>right</i>	<< <i>in</i> >> (p. 1013) The other SampleRejectedStatus to be compared with this status instance. Can be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two SampleRejectedStatus have equal values, or **DDS_BOOLEAN_↵**
FALSE (p. 1201) otherwise.

5.21.5.41 DDS_SubscriptionMatchedStatus_initialize()

```
DDS_ReturnCode_t DDS_SubscriptionMatchedStatus_initialize (
    struct DDS_SubscriptionMatchedStatus * self )
```

Initializer for new status instances.

New **DDS_SubscriptionMatchedStatus** (p. 2022) instances on heap should be initialized with this function before they are passed to any function. This step ensures that those contained fields that use dynamic memory are properly initialized. This function does not allocate memory.

DDS_RequestedIncompatibleQosStatus_finalize (p. 654) should be called to free the contained fields that use dynamic memory:

```
DDS_SubscriptionMatchedStatus *myStatus = malloc(sizeof(struct DDS_SubscriptionMatchedStatus));
```

```
DDS_SubscriptionMatchedStatus_initialize(myStatus);
```

```
...
```

```
DDS_SubscriptionMatchedStatus_finalize(myStatus);
```

```
free(myStatus);
```

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

See also

DDS_SubscriptionMatchedStatus_INITIALIZER (p. 613)

DDS_SubscriptionMatchedStatus_finalize (p. 660)

5.21.5.42 DDS_SubscriptionMatchedStatus_copy()

```
DDS_ReturnCode_t DDS_SubscriptionMatchedStatus_copy (
    struct DDS_SubscriptionMatchedStatus * self,
    const struct DDS_SubscriptionMatchedStatus * source )
```

Copy the contents of the given status into this status.

Status instances can use dynamic memory because of the sequences contained in some status. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). Status to be copied from.

5.21.5.43 DDS_SubscriptionMatchedStatus_finalize()

```
DDS_ReturnCode_t DDS_SubscriptionMatchedStatus_finalize (
    struct DDS_SubscriptionMatchedStatus * self )
```

Free any dynamic memory allocated by status instances.

some status may use dynamic memory (regardless of whether the status itself is in dynamic memory). This function frees that memory but otherwise leaves this status unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

Note that if this status instance is stored in heap memory, calling this function will *not* call free() on it; the user is responsible for explicitly freeing any heap-based status instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.21.5.44 DDS_SubscriptionMatchedStatus_equals()

```
DDS_Boolean DDS_SubscriptionMatchedStatus_equals (
    const struct DDS_SubscriptionMatchedStatus * left,
    const struct DDS_SubscriptionMatchedStatus * right )
```

Compares two **DDS_SubscriptionMatchedStatus** (p. 2022) for equality.

Parameters

<i>left</i>	<< <i>in</i> >> (p. 1013) This SubscriptionMatchedStatus. Can be NULL.
<i>right</i>	<< <i>in</i> >> (p. 1013) The other SubscriptionMatchedStatus to be compared with this status instance. Can be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two SubscriptionMatchedStatus have equal values, or **DDS_BOOLEAN_↵_FALSE** (p. 1201) otherwise.

5.21.5.45 DDS_DataReaderCacheStatus_initialize()

```
DDS_ReturnCode_t DDS_DataReaderCacheStatus_initialize (
    struct DDS_DataReaderCacheStatus * self )
```

Initializer for new status instances.

New **DDS_DataReaderCacheStatus** (p. 1586) instance on heap should be initialized with this function before they are passed to any function. This step ensures that those contained fields that use dynamic memory are properly initialized. This function does not allocate memory.

DDS_DataReaderCacheStatus_finalize (p. 662) should be called to free the contained fields that use dynamic memory:

```
DDS_DataReaderCacheStatus *myStatus = malloc(sizeof(struct DDS_DataReaderCacheStatus));
DDS_DataReaderCacheStatus_initialize(myStatus);
DDS_DataReader_get_datareader_cache_status(myDataReader, myStatus);
DDS_DataReaderCacheStatus_finalize(myStatus);
free(myStatus);
```

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataReader_get_datareader_cache_status (p. 685)

DDS_DataReaderCacheStatus_finalize (p. 662)

5.21.5.46 DDS_DataReaderCacheStatus_copy()

```
DDS_ReturnCode_t DDS_DataReaderCacheStatus_copy (
    struct DDS_DataReaderCacheStatus * self,
    const struct DDS_DataReaderCacheStatus * source )
```

Copy the contents of the given status into this status.

Status instances can use dynamic memory because of the sequences contained in some status. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). Status to be copied from.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataReaderCacheStatus_INITIALIZER (p. 613)

DDS_DataReaderCacheStatus_initialize (p. 661)

DDS_DataReaderCacheStatus_finalize (p. 662)

5.21.5.47 DDS_DataReaderCacheStatus_finalize()

```
DDS_ReturnCode_t DDS_DataReaderCacheStatus_finalize (
    struct DDS_DataReaderCacheStatus * self )
```

Free any dynamic memory allocated by status instances.

some status may use dynamic memory (regardless of whether the status itself is in dynamic memory). This function frees that memory but otherwise leaves this status unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

Note that if this status instance is stored in heap memory, calling this function will *not* call free() on it; the user is responsible for explicitly freeing any heap-based status instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataReaderCacheStatus_INITIALIZER (p. 613)

DDS_DataReaderCacheStatus_initialize (p. 661)

5.21.5.48 DDS_DataReaderCacheStatus_equals()

```
DDS_Boolean DDS_DataReaderCacheStatus_equals (
    const struct DDS_DataReaderCacheStatus * left,
    const struct DDS_DataReaderCacheStatus * right )
```

Compares two **DDS_DataReaderCacheStatus** (p. 1586) for equality.

Parameters

<i>left</i>	<< <i>in</i> >> (p. 1013) This DataReaderCacheStatus. Can be NULL.
<i>right</i>	<< <i>in</i> >> (p. 1013) The other DataReaderCacheStatus to be compared with this status instance. Can be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two DataReaderCacheStatus have equal values, or **DDS_BOOLEAN_↵**
FALSE (p. 1201) otherwise.

5.21.5.49 DDS_DataReaderProtocolStatus_initialize()

```
DDS_ReturnCode_t DDS_DataReaderProtocolStatus_initialize (
    struct DDS_DataReaderProtocolStatus * self )
```

Initializer for new status instances.

New **DDS_DataReaderProtocolStatus** (p. 1600) instance on heap should be initialized with this function before they are passed to any function. This step ensures that those contained fields that use dynamic memory are properly initialized. This function does not allocate memory.

DDS_DataReaderProtocolStatus_finalize (p. 664) should be called to free the contained fields that use dynamic memory:

```
DDS_DataReaderProtocolStatus *myStatus = malloc(sizeof(struct DDS_DataReaderProtocolStatus));

DDS_DataReaderProtocolStatus_initialize(myStatus);

DDS_DataReader_get_datareader_protocol_status(myDataReader, myStatus);

DDS_DataReaderProtocolStatus_finalize(myStatus);

free(myStatus);
```

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataReader_get_datareader_protocol_status (p. 686)

DDS_DataReaderProtocolStatus_finalize (p. 664)

5.21.5.50 DDS_DataReaderProtocolStatus_copy()

```
DDS_ReturnCode_t DDS_DataReaderProtocolStatus_copy (
    struct DDS_DataReaderProtocolStatus * self,
    const struct DDS_DataReaderProtocolStatus * source )
```

Copy the contents of the given status into this status.

Status instances can use dynamic memory because of the sequences contained in some status. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). Status to be copied from.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataReaderProtocolStatus_INITIALIZER (p. 614)

DDS_DataReaderProtocolStatus_initialize (p. 663)

DDS_DataReaderProtocolStatus_finalize (p. 664)

5.21.5.51 DDS_DataReaderProtocolStatus_finalize()

```
DDS_ReturnCode_t DDS_DataReaderProtocolStatus_finalize (
    struct DDS_DataReaderProtocolStatus * self )
```

Free any dynamic memory allocated by status instances.

some status may use dynamic memory (regardless of whether the status itself is in dynamic memory). This function frees that memory but otherwise leaves this status unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

Note that if this status instance is stored in heap memory, calling this function will *not* call `free()` on it; the user is responsible for explicitly freeing any heap-based status instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataReaderProtocolStatus_INITIALIZER (p. 614)

DDS_DataReaderProtocolStatus_initialize (p. 663)

5.21.5.52 DDS_DataReaderProtocolStatus_equals()

```
DDS_Boolean DDS_DataReaderProtocolStatus_equals (
    const struct DDS_DataReaderProtocolStatus * left,
    const struct DDS_DataReaderProtocolStatus * right )
```

Compares two **DDS_DataReaderProtocolStatus** (p. 1600) for equality.

Parameters

<i>left</i>	<< <i>in</i> >> (p. 1013) This DataReaderProtocolStatus. Can be NULL.
<i>right</i>	<< <i>in</i> >> (p. 1013) The other DataReaderProtocolStatus to be compared with this status instance. Can be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two DataReaderProtocolStatus have equal values, or **DDS_BOOLEAN_↵_FALSE** (p. 1201) otherwise.

5.21.5.53 DDS_DataReaderQos_equals()

```
DDS_Boolean DDS_DataReaderQos_equals (
    const struct DDS_DataReaderQos * self,
    const struct DDS_DataReaderQos * other )
```

Compares two **DDS_DataReaderQos** (p. 1611) for equality.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) This DataReaderQos.
<i>other</i>	<< <i>in</i> >> (p. 1013) The other DataReaderQos to be compared with this DataReaderQos.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two Qos objects are equal or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.21.5.54 DDS_DataReaderQos_print()

```
DDS_ReturnCode_t DDS_DataReaderQos_print (
    const struct DDS_DataReaderQos * self )
```

Prints this **DDS_DataReaderQos** (p. 1611) to stdout.

Only the differences between this **DDS_DataReaderQos** (p. 1611) and the documented default are printed. If you wish to print everything regardless, see **DDS_DataReaderQos_to_string_w_params** (p. 668). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1926).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_RETCODE_OK (p. 1222) if no error was encountered.

5.21.5.55 DDS_DataReaderQos_to_string()

```
DDS_ReturnCode_t DDS_DataReaderQos_to_string (
    const struct DDS_DataReaderQos * self,
    char * string,
    DDS_UnsignedLong * string_size )
```

Obtains a string representation of this **DDS_DataReaderQos** (p. 1611).

Only the differences between this **DDS_DataReaderQos** (p. 1611) and the documented default are printed to the string. If you wish to print everything regardless, see **DDS_DataReaderQos_to_string_w_params** (p. 668). The output is formatted according to the default values for **DDS_QosPrintFormat** (p. 1926).

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. 1611) is written to the buffer.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>string</i>	<< <i>out</i> >> (p. 1013) The char buffer that will be used to store the string representation of this DDS_DataReaderQos (p. 1611). If NULL, this function will return the required length of this buffer through the <i>string_size</i> parameter.
<i>string_size</i>	<< <i>inout</i> >> (p. 1013) 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. 1222) if no error was encountered.

See also

DDS_DataReaderQos_to_string_w_params (p. 668)

5.21.5.56 **DDS_DataReaderQos_to_string_w_params()**

```
DDS_ReturnCode_t DDS_DataReaderQos_to_string_w_params (
    const struct DDS_DataReaderQos * self,
    char * string,
    DDS_UnsignedLong * string_size,
    const struct DDS_DataReaderQos * base,
    const struct DDS_QosPrintFormat * format )
```

Obtains a string representation of this **DDS_DataReaderQos** (p. 1611).

Only the differences between this **DDS_DataReaderQos** (p. 1611) and the **DDS_DataReaderQos** (p. 1611) supplied as the base are printed to the string.

It is possible to supply the sentinel value **DDS_DATAREADER_QOS_PRINT_ALL** (p. 600) as the base to print everything within the QoS.

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. 1611) is written to the buffer.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>string</i>	<< <i>out</i> >> (p. 1013) The char buffer that will be used to store the string representation of this DDS_DataReaderQos (p. 1611). If NULL, this function will return the required length of this buffer through the <i>string_size</i> parameter.
<i>string_size</i>	<< <i>inout</i> >> (p. 1013) 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.
<i>base</i>	<< <i>in</i> >> (p. 1013) The DDS_DataReaderQos (p. 1611) to be used as the base QoS profile. Only the differences with respect to this base profile will be included in the output string.
<i>format</i>	<< <i>in</i> >> (p. 1013) The DDS_QosPrintFormat (p. 1926) to be used to format the output string.

Returns

DDS_RETCODE_OK (p. 1222) if no error was encountered.

5.21.5.57 DDS_DataReaderQos_initialize()

```
DDS_ReturnCode_t DDS_DataReaderQos_initialize (
    struct DDS_DataReaderQos * self )
```

Initializer for new QoS instances.

New **DDS_DataReaderQos** (p. 1611) instances on heap should be initialized with this function before they are passed to any function. This step ensures that those contained QoS policies that use dynamic memory are properly initialized. This function does not allocate memory.

Calling this function is not a substitute for calling **DDS_DataReader_get_qos** (p. 688) or **DDS_DomainParticipant_get_default_datareader_qos** (p. 96); one of those should be called subsequently to setting the QoS of any new or existing entity. **DDS_DataReaderQos_finalize** (p. 671) should be called to free the contained QoS policies that use dynamic memory:

```
DDS_DataReaderQos *myQos = malloc(sizeof(struct DDS_DataReaderQos));

DDS_DataReaderQos_initialize(myQos);

DDS_DomainParticipantFactory_get_default_datareader_qos(myFactory, myQos);

DDS_DataReaderQos_set_qos(myDataReader, myQos);

DDS_DataReaderQos_finalize(myQos);

free(myQos);
```

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

See also

DDS_DataReaderQos_INITIALIZER (p. 614)

DDS_DomainParticipant_get_default_datareader_qos (p. 96)

DDS_DataReaderQos_finalize (p. 671)

5.21.5.58 DDS_DataReaderQos_copy()

```
DDS_ReturnCode_t DDS_DataReaderQos_copy (
    struct DDS_DataReaderQos * self,
    const struct DDS_DataReaderQos * source )
```

Copy the contents of the given QoS into this QoS.

DDS_DataReaderQos (p. 1611) instances can use dynamic memory because of the sequences contained in some QoS policies. A shallow copy by assignment is therefore unsafe. This function performs a deep-copy, allocating memory if necessary.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>source</i>	<< <i>in</i> >> (p. 1013). Status to be copied from.

5.21.5.59 DDS_DataReaderQos_finalize()

```
DDS_ReturnCode_t DDS_DataReaderQos_finalize (
    struct DDS_DataReaderQos * self )
```

Free any dynamic memory allocated by the policies in this **DDS_DataReaderQos** (p. 1611).

Some QoS policies may use dynamic memory (regardless of whether the QoS itself is in dynamic memory). This function frees that memory but otherwise leaves this QoS unchanged. It should be called on all instances before they are freed (or, in the case of stack-based instances, before they go out of scope).

This function does not leave this object in an invalid state. It is permissible to clear a QoS and then subsequently allocate new dynamic memory in one or more of its QoS policies.

Note that if this QoS instance is stored in heap memory, calling this function will *not* call free() on it; the user is responsible for explicitly freeing any heap-based QoS instance after calling this function.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.21.5.60 DDS_DataReader_as_entity()

```
DDS_Entity * DDS_DataReader_as_entity (
    DDS_DataReader * dataReader )
```

Accesses the **DDS_DataReader** (p. 615)'s supertype instance.

Parameters

<i>dataReader</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.21.5.61 DDS_DataReader_create_readcondition()

```
DDS_ReadCondition * DDS_DataReader_create_readcondition (
```

```

    DDS_DataReader * self,
    DDS_SampleStateMask sample_states,
    DDS_ViewStateMask view_states,
    DDS_InstanceStateMask instance_states )

```

Creates a **DDS_ReadCondition** (p. 694).

The returned **DDS_ReadCondition** (p. 694) will be attached and belong to the **DDS_DataReader** (p. 615).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>sample_states</i>	<< <i>in</i> >> (p. 1013) sample state of the data samples that are of interest
<i>view_states</i>	<< <i>in</i> >> (p. 1013) view state of the data samples that are of interest
<i>instance_states</i>	<< <i>in</i> >> (p. 1013) instance state of the data samples that are of interest

Returns

return **DDS_ReadCondition** (p. 694) created. Returns NULL in case of failure.

5.21.5.62 DDS_DataReader_create_readcondition_w_params()

```

    DDS_ReadCondition * DDS_DataReader_create_readcondition_w_params (
        DDS_DataReader * self,
        const struct DDS_ReadConditionParams * params )

```

<<*extension*>> (p. 1012) Creates a **DDS_ReadCondition** (p. 694) with parameters.

The returned **DDS_ReadCondition** (p. 694) will be attached and belong to the **DDS_DataReader** (p. 615).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>params</i>	<< <i>in</i> >> (p. 1013) creation parameters.

Returns

return **DDS_ReadCondition** (p. 694) created. Returns NULL in case of failure.

5.21.5.63 DDS_DataReader_create_querycondition()

```

    DDS_QueryCondition * DDS_DataReader_create_querycondition (
        DDS_DataReader * self,

```



```

    DDS_SampleStateMask sample_states,
    DDS_ViewStateMask view_states,
    DDS_InstanceStateMask instance_states,
    const char * query_expression,
    const struct DDS_StringSeq * query_parameters )

```

Creates a **DDS_QueryCondition** (p. 699).

The returned **DDS_QueryCondition** (p. 699) will be attached and belong to the **DDS_DataReader** (p. 615).

Queries and Filters Syntax (p. 738) describes the syntax of `query_expression` and `query_parameters`.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>sample_states</i>	<< <i>in</i> >> (p. 1013) sample state of the data samples that are of interest
<i>view_states</i>	<< <i>in</i> >> (p. 1013) view state of the data samples that are of interest
<i>instance_states</i>	<< <i>in</i> >> (p. 1013) instance state of the data samples that are of interest
<i>query_expression</i>	<< <i>in</i> >> (p. 1013) Expression for the query. Cannot be NULL.
<i>query_parameters</i>	<< <i>in</i> >> (p. 1013) Parameters for the query expression. Cannot be NULL.

Returns

return **DDS_QueryCondition** (p. 699) created. Returns NULL in case of failure.

5.21.5.64 DDS_DataReader_create_querycondition_w_params()

```

DDS_QueryCondition * DDS_DataReader_create_querycondition_w_params (
    DDS_DataReader * self,
    const struct DDS_QueryConditionParams * params )

```

<<*extension*>> (p. 1012) Creates a **DDS_QueryCondition** (p. 699) with parameters.

The returned **DDS_QueryCondition** (p. 699) will be attached and belong to the **DDS_DataReader** (p. 615).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>params</i>	<< <i>in</i> >> (p. 1013) creation parameters.

Returns

return **DDS_QueryCondition** (p. 699) created. Returns NULL in case of failure.

5.21.5.65 DDS_DataReader_delete_readcondition()

```
DDS_ReturnCode_t DDS_DataReader_delete_readcondition (
    DDS_DataReader * self,
    DDS_ReadCondition * condition )
```

Deletes a **DDS_ReadCondition** (p. 694) or **DDS_QueryCondition** (p. 699) attached to the **DDS_DataReader** (p. 615).

Since **DDS_QueryCondition** (p. 699) specializes **DDS_ReadCondition** (p. 694), it can also be used to delete a **DDS_←_QueryCondition** (p. 699).

Precondition

The **DDS_ReadCondition** (p. 694) must be attached to the **DDS_DataReader** (p. 615), or the operation will fail with the error **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>condition</i>	<< <i>in</i> >> (p. 1013) Condition to be deleted.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222)

5.21.5.66 DDS_DataReader_delete_contained_entities()

```
DDS_ReturnCode_t DDS_DataReader_delete_contained_entities (
    DDS_DataReader * self )
```

Deletes all the entities that were created by means of the "create" operations on the **DDS_DataReader** (p. 615).

Deletes all contained **DDS_ReadCondition** (p. 694), **DDS_QueryCondition** (p. 699), and **DDS_TopicQuery** (p. 707) objects.

The operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) if the any of the contained entities is in a state where it cannot be deleted.

Once **DDS_DataReader_delete_contained_entities** (p. 674) completes successfully, the application may delete the **DDS_DataReader** (p. 615).

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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222)

5.21.5.67 DDS_DataReader_wait_for_historical_data()

```
DDS_ReturnCode_t DDS_DataReader_wait_for_historical_data (
    DDS_DataReader * self,
    const struct DDS_Duration_t * max_wait )
```

Waits until all "historical" data is received for **DDS_DataReader** (p. 615) entities that have a non-VOLATILE Durability QoS kind.

This operation is intended only for **DDS_DataReader** (p. 615) entities that have a non-VOLATILE Durability QoS kind.

As soon as an application enables a non-VOLATILE **DDS_DataReader** (p. 615), it will start receiving both "historical" data (i.e., the data that was written prior to the time the **DDS_DataReader** (p. 615) joined the domain) as well as any new data written by the **DDS_DataWriter** (p. 485) 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 **DDS_DataReader_wait_for_historical_data** (p. 675) operations.

DDS_DataReader_wait_for_historical_data() (p. 675) 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 **DDS_DataWriter** (p. 485) has been discovered before calling this operation. (One way to do this is by using **DDS_DataReader_get_subscription_matched_status** (p. 683).)

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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>max_wait</i>	<< <i>in</i> >> (p. 1013) Timeout value. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_TIMEOUT** (p. 1222) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

5.21.5.68 DDS_DataReader_acknowledge_sample_w_response()

```
DDS_ReturnCode_t DDS_DataReader_acknowledge_sample_w_response (
    DDS_DataReader * self,
    const struct DDS_SampleInfo * sample_info,
    const struct DDS_AckResponseData_t * response_data )
```

<<*extension*>> (p. 1012) Acknowledges a single sample explicitly.

Applicable only when **DDS_ReliabilityQosPolicy::acknowledgment_kind** (p.1939) = **DDS_APPLICATION_↵EXPLICIT_ACKNOWLEDGMENT_MODE** (p. 1324)

A call to this function 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. 1959) and **DDS_RtpsReliableReaderProtocol_t::app_ack_↵_period** (p. 1958).

The maximum length of the response is configured using **DDS_DataReaderResourceLimitsQosPolicy::max_app_↵ack_response_length** (p. 1633)

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>sample_info</i>	<< <i>in</i> >> (p. 1013) DDS_SampleInfo (p. 1985) identifying the sample being acknowledged.
<i>response_data</i>	<< <i>in</i> >> (p. 1013) Response data sent to DDS_DataWriter (p. 485) upon acknowledgment (via DDS_DataWriterListener::on_application_acknowledgment (p. 1648))

Returns

One of the **Standard Return Codes** (p. 1221)

5.21.5.69 DDS_DataReader_acknowledge_all_w_response()

```
DDS_ReturnCode_t DDS_DataReader_acknowledge_all_w_response (
    DDS_DataReader * self,
    const struct DDS_AckResponseData_t * response_data )
```

<<*extension*>> (p. 1012) Acknowledges all previously accessed samples.

Applicable only when **DDS_ReliabilityQosPolicy::acknowledgment_kind** (p.1939) = **DDS_APPLICATION_↵EXPLICIT_ACKNOWLEDGMENT_MODE** (p. 1324)

A call to this function 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. 1959) and **DDS_RtpsReliableReaderProtocol_t::app_ack_↵_period** (p. 1958).

The maximum length of the response is configured using **DDS_DataReaderResourceLimitsQosPolicy::max_app_↵ack_response_length** (p. 1633).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>response_data</i>	<< <i>in</i> >> (p. 1013) Response data sent to DDS_DataWriter (p. 485) upon acknowledgment

Returns

One of the **Standard Return Codes** (p. 1221)

5.21.5.70 **DDS_DataReader_acknowledge_sample()**

```
DDS_ReturnCode_t DDS_DataReader_acknowledge_sample (
    DDS_DataReader * self,
    const struct DDS_SampleInfo * sample_info )
```

<<*extension*>> (p. 1012) Acknowledges a single sample explicitly.

Applicable only when **DDS_ReliabilityQosPolicy::acknowledgment_kind** (p. 1939) = **DDS_APPLICATION_↵EXPLICIT_ACKNOWLEDGMENT_MODE** (p. 1324)

A call to this function 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. 1959) and **DDS_RtpsReliableReaderProtocol_t::app_ack_↵period** (p. 1958).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>sample_info</i>	<< <i>in</i> >> (p. 1013) DDS_SampleInfo (p. 1985) identifying the sample being acknowledged.

Returns

One of the **Standard Return Codes** (p. 1221)

5.21.5.71 **DDS_DataReader_acknowledge_all()**

```
DDS_ReturnCode_t DDS_DataReader_acknowledge_all (
    DDS_DataReader * self )
```

<<*extension*>> (p. 1012) Acknowledges all previously accessed samples.

Applicable only when **DDS_ReliabilityQosPolicy::acknowledgment_kind** (p. 1939) = **DDS_APPLICATION_↵EXPLICIT_ACKNOWLEDGMENT_MODE** (p. 1324)

A call to this function 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. 1959) and **DDS_RtpsReliableReaderProtocol_t::app_ack↵_period** (p. 1958).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.21.5.72 DDS_DataReader_get_matched_publications()

```
DDS_ReturnCode_t DDS_DataReader_get_matched_publications (
    DDS_DataReader * self,
    struct DDS_InstanceHandleSeq * publication_handles )
```

Retrieves the list of publications currently "associated" with this **DDS_DataReader** (p. 615).

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 **DDS_Topic** (p. 175).
- The publication has compatible QoS.
- If the applications are using partitions, the publication shares a common partition with this subscription.
- The **DDS_DomainParticipant** (p. 77) has not indicated that the publication's **DDS_DomainParticipant** (p. 77) should be "ignored" by means of the **DDS_DomainParticipant_ignore_publication** (p. 132) API.
- If the subscription is using a **DDS_ContentFilteredTopic** (p. 177) and the publication is using the **DDS_Multi↵ChannelQosPolicy** (p. 1856), 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 **DDS_DataWriter** (p. 485) entities. These handles match the ones that appear in the `instance_handle` field of the **DDS_SampleInfo** (p. 1985) when reading the **DDS_PUBLICATION_↵_TOPIC_NAME** (p. 1096) builtin topic.

This API may return the publication handles of publications that are not alive. **DDS_DataReader_is_matched_↵publication_alive** (p. 679) can be used to check the liveness of the remote publication.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>publication_handles</i>	<< <i>inout</i> >> (p. 1013). 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 function will fail with DDS_RETCODE_OUT_OF_RESOURCES (p. 1222). The maximum number of matches possible is configured with DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731). You can use a zero-maximum sequence without ownership to quickly check whether there are any matches without allocating any memory. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222) if the sequence is too small and the system cannot resize it, or **DDS_RETCODE_NOT_ENABLED** (p. 1222)

5.21.5.73 DDS_DataReader_is_matched_publication_alive()

```
DDS_ReturnCode_t DDS_DataReader_is_matched_publication_alive (
    DDS_DataReader * self,
    DDS_Boolean * is_alive,
    const DDS_InstanceHandle_t * publication_handle )
```

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. 1296) QoS policy was not respected. Note that if the participant associated with the matched publication loses liveliness, the **DDS_InstanceHandle_t** (p. 214) will become invalid and this function will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 1222).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>publication_handle</i>	<< <i>in</i> >> (p. 1013) The DDS_InstanceHandle_t (p. 214) of the matched publication. See DDS_DataReader_get_matched_publications (p. 678) for a description of what is considered a matched publication.
<i>is_alive</i>	<< <i>out</i> >> (p. 1013) Indicates whether or not the matched publication is alive.

Returns

One of the **Standard Return Codes** (p. 1221)

5.21.5.74 DDS_DataReader_get_matched_publication_data()

```
DDS_ReturnCode_t DDS_DataReader_get_matched_publication_data (
    DDS_DataReader * self,
    struct DDS_PublicationBuiltinTopicData * publication_data,
    const DDS_InstanceHandle_t * publication_handle )
```

Retrieves the information on a publication that is currently "associated" with the **DDS_DataReader** (p. 615).

The `publication_handle` must correspond to a publication currently associated with the **DDS_DataReader** (p. 615). Otherwise, the operation will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 1222). Use the operation **DDS_DataReader_get_matched_publications** (p. 678) to find the publications that are currently matched with the **DDS_DataReader** (p. 615).

Note: This operation does not retrieve the **DDS_PublicationBuiltinTopicData::type_code** (p. 1910). This information is available through **DDS_DataReaderListener::on_data_available()** (p. 1595) (if a reader listener is installed on the **DDS_PublicationBuiltinTopicDataDataReader** (p. 1095)).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>publication_data</i>	<< <i>inout</i> >> (p. 1013). The information to be filled in on the associated publication. Cannot be NULL.
<i>publication_handle</i>	<< <i>in</i> >> (p. 1013). Handle to a specific publication associated with the DDS_DataWriter (p. 485). Must correspond to a publication currently associated with the DDS_DataReader (p. 615). Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_NOT_ENABLED** (p. 1222)

5.21.5.75 DDS_DataReader_get_matched_publication_participant_data()

```
DDS_ReturnCode_t DDS_DataReader_get_matched_publication_participant_data (
    DDS_DataReader * self,
    struct DDS_ParticipantBuiltinTopicData * participant_data,
    const DDS_InstanceHandle_t * publication_handle )
```

<<*extension*>> (p. 1012) Retrieves the information on the discovered **DDS_DomainParticipant** (p. 77) associated with the publication that is currently matching with the **DDS_DataReader** (p. 615).

The `publication_handle` must correspond to a publication currently associated with the **DDS_DataReader** (p. 615). Otherwise, the operation will fail with **DDS_RETCODE_BAD_PARAMETER** (p. 1222). The operation may also fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) if the publication corresponds to the same **DDS_DomainParticipant** (p. 77) that the DataReader belongs to. Use the operation **DDS_DataReader_get_matched_publications** (p. 678) to find the publications that are currently matched with the **DDS_DataReader** (p. 615).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>participant_data</i>	<< <i>inout</i> >> (p. 1013). The information to be filled in on the associated participant. Cannot be NULL.
<i>publication_handle</i>	<< <i>in</i> >> (p. 1013). Handle to a specific publication associated with a DDS_DataWriter (p. 485). Must correspond to a publication currently associated with the DDS_DataReader (p. 615). Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_NOT_ENABLED** (p. 1222)

5.21.5.76 DDS_DataReader_get_topicdescription()

```
DDS_TopicDescription * DDS_DataReader_get_topicdescription (
    DDS_DataReader * self )
```

Returns the **DDS_TopicDescription** (p. 175) associated with the **DDS_DataReader** (p. 615).

Returns that same **DDS_TopicDescription** (p. 175) that was used to create the **DDS_DataReader** (p. 615).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_TopicDescription (p. 175) associated with the **DDS_DataReader** (p. 615).

5.21.5.77 DDS_DataReader_get_subscriber()

```
DDS_Subscriber * DDS_DataReader_get_subscriber (
    DDS_DataReader * self )
```

Returns the **DDS_Subscriber** (p. 572) to which the **DDS_DataReader** (p. 615) belongs.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_Subscriber (p. 572) to which the **DDS_DataReader** (p. 615) belongs.

5.21.5.78 DDS_DataReader_get_sample_rejected_status()

```
DDS_ReturnCode_t DDS_DataReader_get_sample_rejected_status (
    DDS_DataReader * self,
    struct DDS_SampleRejectedStatus * status )
```

Accesses the **DDS_SAMPLE_REJECTED_STATUS** (p. 1230) communication status.

This also resets the status so that it is no longer considered changed.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>status</i>	<< <i>inout</i> >> (p. 1013) DDS_SampleRejectedStatus (p. 1997) to be filled in. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.21.5.79 DDS_DataReader_get_liveliness_changed_status()

```
DDS_ReturnCode_t DDS_DataReader_get_liveliness_changed_status (
    DDS_DataReader * self,
    struct DDS_LivelinessChangedStatus * status )
```

Accesses the **DDS_LIVELINESS_CHANGED_STATUS** (p. 1231) communication status.

This also resets the status so that it is no longer considered changed.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>status</i>	<< <i>inout</i> >> (p. 1013) DDS_LivelinessChangedStatus (p. 1820) to be filled in. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.21.5.80 DDS_DataReader_get_requested_deadline_missed_status()

```
DDS_ReturnCode_t DDS_DataReader_get_requested_deadline_missed_status (
    DDS_DataReader * self,
    struct DDS_RequestedDeadlineMissedStatus * status )
```

Accesses the **DDS_REQUESTED_DEADLINE_MISSED_STATUS** (p. 1229) communication status.

This also resets the status so that it is no longer considered changed.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>status</i>	<< <i>inout</i> >> (p. 1013) DDS_RequestedDeadlineMissedStatus (p. 1946) to be filled in. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.21.5.81 DDS_DataReader_get_requested_incompatible_qos_status()

```
DDS_ReturnCode_t DDS_DataReader_get_requested_incompatible_qos_status (
    DDS_DataReader * self,
    struct DDS_RequestedIncompatibleQosStatus * status )
```

Accesses the **DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 1229) communication status.

This also resets the status so that it is no longer considered changed.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>status</i>	<< <i>inout</i> >> (p. 1013) DDS_RequestedIncompatibleQosStatus (p. 1947) to be filled in. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.21.5.82 DDS_DataReader_get_subscription_matched_status()

```
DDS_ReturnCode_t DDS_DataReader_get_subscription_matched_status (
    DDS_DataReader * self,
    struct DDS_SubscriptionMatchedStatus * status )
```

Accesses the **DDS_SUBSCRIPTION_MATCHED_STATUS** (p. 1232) communication status.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>status</i>	<< <i>inout</i> >> (p. 1013) DDS_SubscriptionMatchedStatus (p. 2022) to be filled in. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.21.5.83 DDS_DataReader_get_sample_lost_status()

```
DDS_ReturnCode_t DDS_DataReader_get_sample_lost_status (
    DDS_DataReader * self,
    struct DDS_SampleLostStatus * status )
```

Accesses the **DDS_SAMPLE_LOST_STATUS** (p. 1230) communication status.

This also resets the status so that it is no longer considered changed.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>status</i>	<< <i>inout</i> >> (p. 1013) DDS_SampleLostStatus (p. 1996) to be filled in. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.21.5.84 DDS_DataReader_get_datareader_cache_status()

```
DDS_ReturnCode_t DDS_DataReader_get_datareader_cache_status (
    DDS_DataReader * self,
    struct DDS_DataReaderCacheStatus * status )
```

<<*extension*>> (p. 1012) Gets the datareader cache status for this reader.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>status</i>	<< <i>inout</i> >> (p. 1013) DDS_DataReaderCacheStatus (p. 1586) to be filled in. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

5.21.5.85 DDS_DataReader_get_datareader_protocol_status()

```
DDS_ReturnCode_t DDS_DataReader_get_datareader_protocol_status (
    DDS_DataReader * self,
    struct DDS_DataReaderProtocolStatus * status )
```

<<**extension**>> (p. 1012) Gets the datareader protocol status for this reader.

This also resets the status so that it is no longer considered changed.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>status</i>	<< inout >> (p. 1013) DDS_DataReaderProtocolStatus (p. 1600) to be filled in. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_NOT_ENABLED** (p. 1222).

5.21.5.86 DDS_DataReader_get_matched_publication_datareader_protocol_status()

```
DDS_ReturnCode_t DDS_DataReader_get_matched_publication_datareader_protocol_status (
    DDS_DataReader * self,
    struct DDS_DataReaderProtocolStatus * status,
    const DDS_InstanceHandle_t * publication_handle )
```

<<**extension**>> (p. 1012) 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

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>status</i>	<< inout >> (p. 1013). The information to be filled in on the associated publication. Cannot be NULL.
<i>publication_handle</i>	<< in >> (p. 1013). Handle to a specific publication associated with the DDS_DataWriter (p. 485). Cannot be NULL. . Must correspond to a publication currently associated with the DDS_DataReader (p. 615).
Generated by Doxygen	

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_NOT_ENABLED** (p. 1222)

5.21.5.87 DDS_DataReader_set_qos()

```
DDS_ReturnCode_t DDS_DataReader_set_qos (
    DDS_DataReader * self,
    const struct DDS_DataReaderQos * qos )
```

Sets the reader QoS.

Modifies the QoS of the **DDS_DataReader** (p. 615).

The **DDS_DataReaderQos::user_data** (p. 1614), **DDS_DataReaderQos::deadline** (p. 1613), **DDS_DataReaderQos::latency_budget** (p. 1613), **DDS_DataReaderQos::time_based_filter** (p. 1615), **DDS_DataReaderQos::reader_data_lifecycle** (p. 1615) can be changed. The other policies are immutable.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>in</i> >> (p. 1013) The DDS_DataReaderQos (p. 1611) to be set to. Policies must be consistent. Immutable policies cannot be changed after DDS_DataReader (p. 615) is enabled. The special value DDS_DATAREADER_QOS_DEFAULT (p. 600) can be used to indicate that the QoS of the DDS_DataReader (p. 615) should be changed to match the current default DDS_DataReaderQos (p. 1611) set in the DDS_Subscriber (p. 572). Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 1222), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222).

See also

DDS_DataReaderQos (p. 1611) for rules on consistency among QoS

set_qos (abstract) (p. 1361)

DDS_DataReader_set_qos (p. 687)

Operations Allowed in Listener Callbacks (p. 1818)

5.21.5.88 DDS_DataReader_set_qos_with_profile()

```
DDS_ReturnCode_t DDS_DataReader_set_qos_with_profile (
    DDS_DataReader * self,
    const char * library_name,
    const char * profile_name )
```

<<**extension**>> (p. 1012) Changes the QoS of this reader using the input XML QoS profile.

This operation modifies the QoS of the **DDS_DataReader** (p. 615).

The **DDS_DataReaderQos::user_data** (p. 1614), **DDS_DataReaderQos::deadline** (p. 1613), **DDS_DataReaderQos::latency_budget** (p. 1613), **DDS_DataReaderQos::time_based_filter** (p. 1615), **DDS_DataReaderQos::reader_data_lifecycle** (p. 1615) can be changed. The other policies are immutable.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>library_name</i>	<< in >> (p. 1013) Library name containing the XML QoS profile. If library_name is null RTI Connexnt will use the default library (see DDS_Subscriber_set_default_library (p. 595)).
<i>profile_name</i>	<< in >> (p. 1013) XML QoS Profile name. If profile_name is null RTI Connexnt will use the default profile (see DDS_Subscriber_set_default_profile (p. 594)).

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_IMMUTABLE_POLICY** (p. 1222), or **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222).

See also

DDS_DataReaderQos (p. 1611) for rules on consistency among QoS

DDS_DataReader_set_qos (p. 687)

Operations Allowed in Listener Callbacks (p. 1818)

5.21.5.89 DDS_DataReader_get_qos()

```
DDS_ReturnCode_t DDS_DataReader_get_qos (
    DDS_DataReader * self,
    struct DDS_DataReaderQos * qos )
```

Gets the reader QoS.

This function may potentially allocate memory depending on the sequences contained in some QoS policies.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>qos</i>	<< <i>inout</i> >> (p. 1013) The DDS_DataReaderQos (p. 1611) to be filled up. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

get_qos (abstract) (p. 1362)

5.21.5.90 DDS_DataReader_set_property()

```
DDS_ReturnCode_t DDS_DataReader_set_property (
    DDS_DataReader * self,
    const char * property_name,
    const char * value,
    DDS_Boolean propagate )
```

Set the value for a property that applies to a DataReader.

Warning

This function is not implemented in all APIs and it's intended only for testing purposes. You should use **DDS_↔ DataReader_set_qos** (p. 687) instead.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>property_name</i>	<< <i>in</i> >> (p. 1013). Name of the property that you want to set.
<i>value</i>	<< <i>in</i> >> (p. 1013). New value for the property.
<i>propagate</i>	<< <i>in</i> >> (p. 1013). Indicates if the property will be propagated or not.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DomainParticipant_set_property (p. 153)

DDS_DataWriter_set_property (p. 552)

DDS_DataReader_set_qos (p. 687)

5.21.5.91 DDS_DataReader_set_listener()

```
DDS_ReturnCode_t DDS_DataReader_set_listener (
    DDS_DataReader * self,
    const struct DDS_DataReaderListener * l,
    DDS_StatusMask mask )
```

Sets the reader listener.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>l</i>	<< <i>in</i> >> (p. 1013) DDS_DataReaderListener (p. 1593) to set to
<i>mask</i>	<< <i>in</i> >> (p. 1013) DDS_StatusMask (p. 1226) associated with the DDS_DataReaderListener (p. 1593). The callback function on the listener cannot be NULL if the corresponding status is turned on in the <i>mask</i> .

Returns

One of the **Standard Return Codes** (p. 1221)

See also

set_listener (abstract) (p. 1362)

5.21.5.92 DDS_DataReader_get_listener()

```
struct DDS_DataReaderListener DDS_DataReader_get_listener (
    DDS_DataReader * self )
```

Gets the reader listener.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_DataReaderListener (p. 1593) of the **DDS_DataReader** (p. 615).

See also

DDS_DataReader_get_listenerX (p. 691)

get_listener (abstract) (p. 1363)

5.21.5.93 DDS_DataReader_get_listenerX()

```
DDS_ReturnCode_t DDS_DataReader_get_listenerX (
    DDS_DataReader * self,
    struct DDS_DataReaderListener * listener )
```

<<**extension**>> (p. 1012) Gets the reader listener.

An alternative form of **get_listener** that fills in an existing listener structure rather than returning one on the stack.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>listener</i>	<< inout >> (p. 1013) DDS_DataReaderListener (p. 1593) structure to be filled up. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_DataReader_get_listener (p. 690)

get_listener (abstract) (p. 1363)

5.21.5.94 DDS_DataReader_create_topic_query()

```
DDS_TopicQuery * DDS_DataReader_create_topic_query (
    DDS_DataReader * self,
    const struct DDS_TopicQuerySelection * selection )
```

Creates a **DDS_TopicQuery** (p. 707).

The returned **DDS_TopicQuery** (p. 707) will have been issued if the **DDS_DataReader** (p. 615) is enabled. Otherwise, the **DDS_TopicQuery** (p. 707) will be issued once the **DDS_DataReader** (p. 615) is enabled

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>selection</i>	<< <i>in</i> >> (p. 1013) The selection with which to create the DDS_TopicQuery (p. 707). The special values DDS_TOPIC_QUERY_SELECTION_SELECT_ALL (p. 709) and DDS_TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER (p. 709) 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. Cannot be NULL.

Returns

return Created **DDS_TopicQuery** (p. 707). Returns NULL in case of failure.

5.21.5.95 DDS_DataReader_delete_topic_query()

```
DDS_ReturnCode_t DDS_DataReader_delete_topic_query (
    DDS_DataReader * self,
    DDS_TopicQuery * query )
```

Deletes a **DDS_TopicQuery** (p. 707).

Cancels an active **DDS_TopicQuery** (p. 707). 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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>query</i>	<< <i>in</i> >> (p. 1013) The DDS_TopicQuery (p. 707) to delete. Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.21.5.96 DDS_DataReader_lookup_topic_query()

```
DDS_TopicQuery * DDS_DataReader_lookup_topic_query (
    DDS_DataReader * self,
    const struct DDS_GUID_t * guid )
```

Retrieves an existing **DDS_TopicQuery** (p. 707).

Retrieves the **DDS_TopicQuery** (p. 707) that corresponds to the input **DDS_GUID_t** (p. 1804).

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. 1804) associated with a **DDS_TopicQuery** (p. 707), use the function **DDS_TopicQuery_↔get_guid** (p. 708).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>guid</i>	<< <i>in</i> >> (p. 1013) The DDS_TopicQuery (p. 707) GUID. Cannot be NULL.

5.21.5.97 DDS_DataReader_take_discovery_snapshot()

```
DDS_ReturnCode_t DDS_DataReader_take_discovery_snapshot (
    DDS_DataReader * self,
    const char * file_name )
```

Take a snapshot of the compatible and incompatible remote writers matched by a local reader.

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="udp4://192.168.1.170:7411"

liveliness="ALIVE"

Incompatible writers:

1. 0x0101D8D1,0x20B83C0D,0x4495246E:0x80000103 name="writer2TestName"

kind="unkeyed user datareader"

unicastLocators="udp4://192.168.1.170:7411"

reason="Inconsistent QoS"

Precondition

self is not NULL.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) The local reader.
<i>file_name</i>	<< <i>in</i> >> (p. 1013) Name of the file where snapshot should be printed. If NULL, the snapshot will be printed through NDDS_Config_Logger (p. 2116). Otherwise, the snapshot will be printed in the file specified by <i>file_name</i> .

Returns

One of the **Standard Return Codes** (p. 1221).

5.22 Read Conditions

DDS_ReadCondition (p. 694) and associated elements

Data Structures

- struct **DDS_ReadConditionParams**
*<<extension>> (p. 1012) Input parameters for **DDS_DataReader_create_readcondition_w_params** (p. 672)*

Typedefs

- typedef struct DDS_ReadConditionImpl **DDS_ReadCondition**
*<<interface>> (p. 1013) Conditions specifically dedicated to read operations and attached to one **DDS_DataReader** (p. 615).*

Functions

- **DDS_Condition * DDS_ReadCondition_as_condition (DDS_ReadCondition *read_condition)**
*Accesses a **DDS_ReadCondition** (p. 694)'s supertype instance.*
- **DDS_SampleStateMask DDS_ReadCondition_get_sample_state_mask (DDS_ReadCondition *self)**
*Retrieves the set of *sample_states* for the condition.*
- **DDS_ViewStateMask DDS_ReadCondition_get_view_state_mask (DDS_ReadCondition *self)**
*Retrieves the set of *view_states* for the condition.*
- **DDS_InstanceStateMask DDS_ReadCondition_get_instance_state_mask (DDS_ReadCondition *self)**
*Retrieves the set of *instance_states* for the condition.*
- **DDS_StreamKindMask DDS_ReadCondition_get_stream_kind_mask (DDS_ReadCondition *self)**
*Retrieves the set of *stream kind mask* for the condition.*
- **DDS_DataReader * DDS_ReadCondition_get_datareader (DDS_ReadCondition *self)**
*Returns the **DDS_DataReader** (p. 615) associated with the **DDS_ReadCondition** (p. 694).*

5.22.1 Detailed Description

DDS_ReadCondition (p. 694) and associated elements

5.22.2 Typedef Documentation

5.22.2.1 DDS_ReadCondition

```
typedef struct DDS_ReadConditionImpl DDS_ReadCondition
```

<<*interface*>> (p. 1013) Conditions specifically dedicated to read operations and attached to one **DDS_DataReader** (p. 615).

DDS_ReadCondition (p. 694) 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. 625) and **FooDataReader_take** (p. 626) 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 **DDS_ReadCondition** (p. 694) may be attached to the same **DDS_DataReader** (p. 615).

5.22.3 Function Documentation

5.22.3.1 DDS_ReadCondition_as_condition()

```
DDS_Condition * DDS_ReadCondition_as_condition (
    DDS_ReadCondition * read_condition )
```

Accesses a **DDS_ReadCondition** (p. 694)'s supertype instance.

Parameters

<i>read_condition</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.22.3.2 DDS_ReadCondition_get_sample_state_mask()

```
DDS_SampleStateMask DDS_ReadCondition_get_sample_state_mask (
    DDS_ReadCondition * self )
```

Retrieves the set of `sample_states` for the condition.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.22.3.3 DDS_ReadCondition_get_view_state_mask()

```
DDS_ViewStateMask DDS_ReadCondition_get_view_state_mask (
    DDS_ReadCondition * self )
```

Retrieves the set of `view_states` for the condition.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.22.3.4 DDS_ReadCondition_get_instance_state_mask()

```
DDS_InstanceStateMask DDS_ReadCondition_get_instance_state_mask (
    DDS_ReadCondition * self )
```

Retrieves the set of `instance_states` for the condition.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.22.3.5 DDS_ReadCondition_get_stream_kind_mask()

```
DDS_StreamKindMask DDS_ReadCondition_get_stream_kind_mask (
    DDS_ReadCondition * self )
```

Retrieves the set of stream kind mask for the condition.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.22.3.6 DDS_ReadCondition_get_datareader()

```
DDS_DataReader * DDS_ReadCondition_get_datareader (
    DDS_ReadCondition * self )
```


Returns the **DDS_DataReader** (p. 615) associated with the **DDS_ReadCondition** (p. 694).

There is exactly one **DDS_DataReader** (p. 615) associated with each **DDS_ReadCondition** (p. 694).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_DataReader (p. 615) associated with the **DDS_ReadCondition** (p. 694).

5.23 Query Conditions

DDS_QueryCondition (p. 699) and associated elements

Data Structures

- struct **DDS_QueryConditionParams**
 <<*extension*>> (p. 1012) Input parameters for **DDS_DataReader_create_querycondition_w_params** (p. 673)

Typedefs

- typedef struct DDS_QueryConditionImpl **DDS_QueryCondition**
 <<*interface*>> (p. 1013) These are specialised **DDS_ReadCondition** (p. 694) objects that allow the application to also specify a filter on the locally available data.

Functions

- **DDS_ReadCondition * DDS_QueryCondition_as_readcondition** (**DDS_QueryCondition** *query_condition)
 Access a **DDS_QueryCondition** (p. 699)'s supertype instance.
- const char * **DDS_QueryCondition_get_query_expression** (**DDS_QueryCondition** *self)
 Retrieves the query expression.
- **DDS_ReturnCode_t DDS_QueryCondition_get_query_parameters** (**DDS_QueryCondition** *self, struct **DDS_StringSeq** *query_parameters)
 Retrieves the query parameters.
- **DDS_ReturnCode_t DDS_QueryCondition_set_query_parameters** (**DDS_QueryCondition** *self, const struct **DDS_StringSeq** *query_parameters)
 Sets the query parameters.

5.23.1 Detailed Description

DDS_QueryCondition (p. 699) and associated elements

5.23.2 Typedef Documentation

5.23.2.1 DDS_QueryCondition

```
typedef struct DDS_QueryConditionImpl DDS_QueryCondition
```

<<*interface*>> (p. 1013) These are specialised **DDS_ReadCondition** (p. 694) objects that allow the application to also specify a filter on the locally available data.

Each query condition filter is composed of a **DDS_ReadCondition** (p. 694) 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()` 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. 738) describes the syntax of `query_expression` and `query_parameters`.

5.23.3 Function Documentation

5.23.3.1 DDS_QueryCondition_as_readcondition()

```
DDS_ReadCondition * DDS_QueryCondition_as_readcondition (
    DDS_QueryCondition * query_condition )
```

Access a **DDS_QueryCondition** (p. 699)'s supertype instance.

Parameters

<code>query_condition</code>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.23.3.2 DDS_QueryCondition_get_query_expression()

```
const char * DDS_QueryCondition_get_query_expression (
    DDS_QueryCondition * self )
```

Retrieves the query expression.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.23.3.3 DDS_QueryCondition_get_query_parameters()

```
DDS_ReturnCode_t DDS_QueryCondition_get_query_parameters (
    DDS_QueryCondition * self,
    struct DDS_StringSeq * query_parameters )
```

Retrieves the query parameters.

Parameters

<i>query_parameters</i>	<< <i>inout</i> >> (p. 1013) 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. 1527). In particular, be careful to avoid a situation in which RTI Connexx allocates a string on your behalf and you then reuse that string in such a way that RTI Connexx believes it to have more memory allocated to it than it actually does.
<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.23.3.4 DDS_QueryCondition_set_query_parameters()

```
DDS_ReturnCode_t DDS_QueryCondition_set_query_parameters (
    DDS_QueryCondition * self,
    const struct DDS_StringSeq * query_parameters )
```

Sets the query parameters.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>query_parameters</i>	<< <i>in</i> >> (p. 1013) the new query parameters

5.24 Data Samples

DDS_SampleInfo (p. 1985), **DDS_SampleStateKind** (p. 710), **DDS_ViewStateKind** (p. 712), **DDS_InstanceStateKind** (p. 714) and associated elements

Modules

- **Sample States**
DDS_SampleStateKind (p. 710) and associated elements
- **View States**
DDS_ViewStateKind (p. 712) and associated elements
- **Instance States**
DDS_InstanceStateKind (p. 714) and associated elements
- **Stream Kinds**
DDS_StreamKind (p. 716) and associated elements

Data Structures

- struct **DDS_CoherentSetInfo_t**
<<extension>> (p. 1012) 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. 1985) > .

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_CoherentSetInfo_t** *other)
Copies another CoherentSetInfo into this CoherentSetInfo.
- void **DDS_SampleInfo_get_sample_identity** (const struct **DDS_SampleInfo** *self, struct **DDS_SampleIdentity_t** *identity)
<<extension>> (p. 1012) 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. 1012) Retrieves the identity of a sample related to this one

5.24.1 Detailed Description

DDS_SampleInfo (p. 1985), **DDS_SampleStateKind** (p. 710), **DDS_ViewStateKind** (p. 712), **DDS_InstanceStateKind** (p. 714) and associated elements

5.24.2 Function Documentation

5.24.2.1 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

<i>self</i>	<< <i>in</i> >> (p. 1013) This CoherentSetInfo. Cannot be NULL.
<i>other</i>	<< <i>in</i> >> (p. 1013) The other CoherentSetInfo to be compared with this GUID. Cannot be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two CoherentSetInfos have equal values, or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.24.2.2 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

<i>self</i>	<< <i>in</i> >> (p. 1013) This CoherentSetInfo. Cannot be NULL.
<i>other</i>	<< <i>in</i> >> (p. 1013) The other CoherentSetInfo to be copied.

5.24.2.3 DDS_SampleInfo_get_sample_identity()

```
void DDS_SampleInfo_get_sample_identity (
    const struct    DDS_SampleInfo * self,
    struct    DDS_SampleIdentity_t * identity )
```

<<*extension*>> (p. 1012) Retrieves the identity of the sample

The identity is composed of the **DDS_SampleInfo::original_publication_virtual_guid** (p.1993) and the **DDS_SampleInfo::original_publication_virtual_sequence_number** (p.1993)

See also

DDS_SampleInfo_get_related_sample_identity (p. 702)

5.24.2.4 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. 1012) Retrieves the identity of a sample related to this one

A sample can be logically related to another sample, when the **DDS_DataWriter** (p. 485) wrote it using **DDS_WriteParams_t::related_sample_identity** (p. 2102). By default, a sample does not have a related sample, and this operation returns **DDS_UNKNOWN_SAMPLE_IDENTITY** (p. 1392).

The related identity is composed of the **DDS_SampleInfo::related_original_publication_virtual_guid** (p. 1993) and the **DDS_SampleInfo::related_original_publication_virtual_sequence_number** (p. 1994)

See also

DDS_WriteParams_t::related_sample_identity (p. 2102)

5.25 Topic Queries

DDS_TopicQuery (p. 707) and associated elements.

Data Structures

- struct **DDS_TopicQuerySelection**
 <<*extension*>> (p. 1012) Specifies the data query that defines a **DDS_TopicQuery** (p. 707)
- struct **DDS_TopicQueryData**
 <<*extension*>> (p. 1012) Provides information about a **DDS_TopicQuery** (p. 707)

Typedefs

- typedef struct **DDS_TopicQuerySelection DDS_TopicQuerySelection**
 <<*extension*>> (p. 1012) Specifies the data query that defines a **DDS_TopicQuery** (p. 707)
- typedef struct **DDS_TopicQueryImpl DDS_TopicQuery**
 <<*extension*>> (p. 1012) Allows a **DDS_DataReader** (p. 615) to query the sample cache of its matching **DDS_DataWriter** (p. 485).
- typedef struct **DDS_TopicQueryData DDS_TopicQueryData**
 <<*extension*>> (p. 1012) Provides information about a **DDS_TopicQuery** (p. 707)

Enumerations

- enum **DDS_TopicQuerySelectionKind** {
 DDS_TOPIC_QUERY_SELECTION_KIND_HISTORY_SNAPSHOT ,
 DDS_TOPIC_QUERY_SELECTION_KIND_CONTINUOUS }
 Kinds of TopicQuerySelection.

Functions

- **DDS_Boolean DDS_TopicQueryHelper_topic_query_data_from_service_request** (struct **DDS_TopicQueryData** *topic_query_data, const struct **DDS_ServiceRequest** *service_request)
Retrieves the **DDS_TopicQueryData** (p. 2046) data contained in the specified **DDS_ServiceRequest** (p. 2001).
- **DDS_ReturnCode_t DDS_TopicQuery_get_guid** (**DDS_TopicQuery** *self, **DDS_GUID_t** *guid)
Get the *TopicQuery*'s GUID.

Variables

- const struct **DDS_TopicQuerySelection DDS_TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER**
Special value for creating a **DDS_TopicQuery** (p. 707) that applies the same filter as the *DataReader*'s **DDS_ContentFilteredTopic** (p. 177).
- const struct **DDS_TopicQuerySelection DDS_TOPIC_QUERY_SELECTION_SELECT_ALL**
Special value for creating a **DDS_TopicQuery** (p. 707) that selects all the samples in a **DDS_DataWriter** (p. 485) cache.

5.25.1 Detailed Description

DDS_TopicQuery (p. 707) and associated elements.

TopicQueries allow a **DDS_DataReader** (p. 615) to query the sample cache of its matching **DDS_DataWriter** (p. 485). A user can create a **DDS_TopicQuery** (p. 707) with the **DDS_DataReader_create_topic_query** (p. 691) 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. 1666) includes the **DDS_TopicQueryDispatchQosPolicy** (p. 2047) 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 *TopicQuery* 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 **DDS_DataReader_delete_topic_query** (p. 692).

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. 1995) 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. 1213). 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 **DDS_DataReader_create_querycondition_w_params** (p. 673) and **DDS_DataReader_create_readcondition_w_params** (p. 672) APIs.

Each *TopicQuery* is identified by a GUID that can be accessed using the **DDS_TopicQuery_get_guid** (p. 708) method.

5.25.2 Snapshot TopicQueries

A snapshot TopicQuery queries the samples that are in a DataWriter's queue when the DataWriter receives the TopicQuery.

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. 1388) bit in the **DDS_SampleInfo::flag** (p. 1994) 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. 1293) 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. 1388) flag in the **DDS_SampleInfo::flag** (p. 1994) field of the samples included in the response. For more information on how to use this flag, see **Reading TopicQuery Samples** (p. 705).

5.25.3 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>` to PROPAGATION. These DataWriters forward a TopicQuery response, but they are not considered to be responding to the TopicQuery.

Samples that a DataReader receives in response to a TopicQuery have a value for **DDS_SampleInfo::topic_query_guid** (p. 1995) different than **DDS_GUID_UNKNOWN** (p. 1213).

In addition, the virtual GUID of the DataWriter(s) that responded to a TopicQuery is set in **DDS_SampleInfo::source_guid** (p. 1994).

Intermediate samples, not the final sample, have the flag **DDS_INTERMEDIATE_TOPIC_QUERY_SAMPLE** (p. 1388) set in **DDS_SampleInfo::flag** (p. 1994) 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. 1994) responding to the TopicQuery, assuming that at least one sample passes the TopicQuery filter for each DataWriter.

The **DDS_SampleInfo::flag** (p. 1994) field of the last sample received for a snapshot TopicQuery may have the **DDS_INCOMPLETE_SNAPSHOT_TOPIC_QUERY** (p. 1388) bit set if the TopicQuery response is incomplete, which may or may not be necessary for your application to detect. See **Snapshot TopicQueries** (p. 705) for more information.

5.25.4 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.

5.25.4.1 The Built-in ServiceRequest DataReader

TopicQueries are communicated to publishing applications through a built-in **DDS_ServiceRequest** (p. 2001) 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. 2001) message is sent containing the TopicQuery information. Just as there are built-in DataReaders for **DDS_ParticipantBuiltinTopicData** (p. 1870), **DDS_SubscriptionBuiltinTopicData** (p. 2012), and **DDS_PublicationBuiltinTopicData** (p. 1905), there is a fourth built-in DataReader for ServiceRequests.

The new built-in DataReader can be retrieved using the built-in subscriber and **DDS_Subscriber_lookup_datareader** (p. 586). The topic name is **DDS_SERVICE_REQUEST_TOPIC_NAME** (p. 1101). Installing a listener with the **DDS_DataReaderListener::on_data_available** (p. 1595) 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. 2001) of a **DDS_ServiceRequest** (p. 2001) corresponding to a **DDS_TopicQuery** (p. 707) will be **DDS_TOPIC_QUERY_SERVICE_REQUEST_ID** (p. 1101) and the **DDS_ServiceRequest::instance_id** (p. 2001) will be equal to the GUID of the **DDS_TopicQuery** (p. 707) (see **DDS_TopicQuery::get_guid** (p. 708)).

The **DDS_ServiceRequest::request_body** (p. 2002) is a sequence of bytes containing more information about the TopicQuery. This information can be retrieved using the **DDS_TopicQueryHelper_topic_query_data_from_service_request** (p. 708) function. The resulting **DDS_TopicQueryData** (p. 2046) contains the **DDS_TopicQuerySelection** (p. 2049) that the **DDS_TopicQuery** (p. 707) 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. 2047), however, will be that of the first DataReader to have created the TopicQuery.

5.25.4.2 The on_service_request_accepted DataWriter Listener Callback

It is possible that a **DDS_ServiceRequest** (p. 2001) for a **DDS_TopicQuery** (p. 707) 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 **DDS_DataWriterListener::on_service_request_accepted** (p. 1648) callback notifies a DataWriter when a ServiceRequest has been dispatched to that DataWriter. The **DDS_ServiceRequestAcceptedStatus** (p. 2002) 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. 2003), which is the **DDS_InstanceHandle_t** (p. 214) 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.

5.25.5 Typedef Documentation

5.25.5.1 DDS_TopicQuerySelection

```
typedef struct DDS_TopicQuerySelection DDS_TopicQuerySelection
```

<<**extension**>> (p. 1012) Specifies the data query that defines a **DDS_TopicQuery** (p. 707)

The query format is similar to the expression and parameters of a **DDS_QueryCondition** (p. 699) or a **DDS_ContentFilteredTopic** (p. 177), as described in **DDS_DomainParticipant_create_contentfilteredtopic_with_filter** (p. 119).

There are two special queries:

- **DDS_TOPIC_QUERY_SELECTION_SELECT_ALL** (p. 709)
- **DDS_TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER** (p. 709)

See also

Queries and Filters Syntax (p. 738)

5.25.5.2 DDS_TopicQuery

```
typedef struct DDS_TopicQueryImpl DDS_TopicQuery
```

<<**extension**>> (p. 1012) Allows a **DDS_DataReader** (p. 615) to query the sample cache of its matching **DDS_DataWriter** (p. 485).

5.25.5.3 DDS_TopicQueryData

```
typedef struct DDS_TopicQueryData DDS_TopicQueryData
```

<<**extension**>> (p. 1012) Provides information about a **DDS_TopicQuery** (p. 707)

Contains the information about a TopicQuery that can be retrieved using **DDS_TopicQueryHelper_topic_query_data_from_service_request** (p. 708).

See also

DDS_TopicQueryHelper_topic_query_data_from_service_request (p. 708)

5.25.6 Enumeration Type Documentation

5.25.6.1 DDS_TopicQuerySelectionKind

```
enum DDS_TopicQuerySelectionKind
```

Kinds of TopicQuerySelection.

See also

DDS_TopicQuerySelection (p. 2049)

Enumerator

DDS_TOPIC_QUERY_SELECTION_KIND_↔ HISTORY_SNAPSHOT	Indicates that the DDS_TopicQuery (p. 707) may only select samples that were in the DataWriter cache upon reception. [default]
DDS_TOPIC_QUERY_SELECTION_KIND_↔ CONTINUOUS	Indicates that the DDS_TopicQuery (p. 707) may continue selecting samples published after its reception. The subscribing application must explicitly delete the TopicQuery (see DDS_DataReader_delete_topic_query (p. 692)) to signal the DataWriters to stop publishing samples for it.

5.25.7 Function Documentation

5.25.7.1 DDS_TopicQueryHelper_topic_query_data_from_service_request()

```

DDS_Boolean DDS_TopicQueryHelper_topic_query_data_from_service_request (
    struct DDS_TopicQueryData * topic_query_data,
    const struct DDS_ServiceRequest * service_request )

```

Retrieves the **DDS_TopicQueryData** (p. 2046) data contained in the specified **DDS_ServiceRequest** (p. 2001).

This operation will extract the content from the request body of the **DDS_ServiceRequest** (p. 2001) and place it into the specified **DDS_TopicQueryData** (p. 2046) object.

The specified **DDS_ServiceRequest** (p. 2001) must be a valid sample associated with the service id **DDS_TOPIC_↔
QUERY_SERVICE_REQUEST_ID** (p. 1101). Otherwise this operation will return **DDS_BOOLEAN_FALSE** (p. 1201).

This operation can be called within the context of a **DDS_DataWriterListener::on_service_request_accepted** (p. 1648) to retrieve the **DDS_TopicQueryData** (p. 2046) of a **DDS_ServiceRequest** (p. 2001) that has been received with service id **DDS_TOPIC_QUERY_SERVICE_REQUEST_ID** (p. 1101)

Parameters

<i>topic_query_data</i>	<< <i>inout</i> >> (p. 1013) A DDS_TopicQueryData (p. 2046) object where the content from the service request is extracted. Cannot be NULL.
<i>service_request</i>	<< <i>in</i> >> (p. 1013) Input DDS_ServiceRequest (p. 2001) that contains the DDS_TopicQueryData (p. 2046) as part of its request body. Cannot be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) on success.

5.25.7.2 DDS_TopicQuery_get_guid()

```
DDS_ReturnCode_t DDS_TopicQuery_get_guid (
    DDS_TopicQuery * self,
    DDS_GUID_t * guid )
```

Get the TopicQuery's GUID.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>guid</i>	<< <i>out</i> >> (p. 1013) The GUID of the DDS_TopicQuery (p. 707). Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221)

5.25.8 Variable Documentation

5.25.8.1 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 **DDS_TopicQuery** (p. 707) that applies the same filter as the DataReader's **DDS_ContentFilteredTopic** (p. 177).

If the **DDS_DataReader** (p. 615) that creates the **DDS_TopicQuery** (p. 707) uses a **DDS_ContentFilteredTopic** (p. 177), this **DDS_TopicQuerySelection** (p. 2049) value indicates that the TopicQuery should use the same filter.

Otherwise it behaves as **DDS_TOPIC_QUERY_SELECTION_SELECT_ALL** (p. 709).

5.25.8.2 DDS_TOPIC_QUERY_SELECTION_SELECT_ALL

```
const struct DDS_TopicQuerySelection DDS_TOPIC_QUERY_SELECTION_SELECT_ALL [extern]
```

Special value for creating a **DDS_TopicQuery** (p. 707) that selects all the samples in a **DDS_DataWriter** (p. 485) cache.

5.26 Sample States

DDS_SampleStateKind (p. 710) and associated elements

Typedefs

- typedef **DDS_UnsignedLong** **DDS_SampleStateMask**
A bit-mask (list) of sample states, i.e. **DDS_SampleStateKind** (p. 710).

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. 711) | **DDS_NOT_READ_SAMPLE_STATE** (p. 711).

5.26.1 Detailed Description

DDS_SampleStateKind (p. 710) and associated elements

5.26.2 Typedef Documentation

5.26.2.1 DDS_SampleStateMask

```
typedef DDS_UnsignedLong DDS_SampleStateMask
```

A bit-mask (list) of sample states, i.e. **DDS_SampleStateKind** (p. 710).

5.26.3 Enumeration Type Documentation

5.26.3.1 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 **DDS_DataReader** (p. 615). The sample state can be either:

- **DDS_READ_SAMPLE_STATE** (p. 711) indicates that the **DDS_DataReader** (p. 615) has already accessed that sample by means of a read or take operation.
- **DDS_NOT_READ_SAMPLE_STATE** (p. 711) indicates that the **DDS_DataReader** (p. 615) 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.

5.26.4 Variable Documentation

5.26.4.1 DDS_ANY_SAMPLE_STATE

```
const DDS_SampleStateMask DDS_ANY_SAMPLE_STATE [extern]
```

Any sample state **DDS_READ_SAMPLE_STATE** (p. 711) | **DDS_NOT_READ_SAMPLE_STATE** (p. 711).

Examples

HelloWorld_subscriber.c.

5.27 View States

DDS_ViewStateKind (p. 712) and associated elements

Typedefs

- typedef **DDS_UnsignedLong DDS_ViewStateMask**
A bit-mask (list) of view states, i.e. **DDS_ViewStateKind** (p. 712).

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. 713) | **DDS_NOT_NEW_VIEW_STATE** (p. 713).

5.27.1 Detailed Description

DDS_ViewStateKind (p. 712) and associated elements

5.27.2 Typedef Documentation

5.27.2.1 DDS_ViewStateMask

```
typedef DDS_UnsignedLong DDS_ViewStateMask
```

A bit-mask (list) of view states, i.e. **DDS_ViewStateKind** (p. 712).

5.27.3 Enumeration Type Documentation

5.27.3.1 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 **DDS_DataReader** (p. 615). The view state can be either:

- **DDS_NEW_VIEW_STATE** (p. 713) indicates that either this is the first time that the **DDS_DataReader** (p. 615) has ever accessed samples of that instance, or else that the **DDS_DataReader** (p. 615) 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. 1990) and the **DDS_SampleInfo::no_writers_generation_count** (p. 1990).
- **DDS_NOT_NEW_VIEW_STATE** (p. 713) indicates that the **DDS_DataReader** (p. 615) 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. 1985) is a snapshot of the view state of the instance relative to the **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615), 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.

5.27.4 Variable Documentation

5.27.4.1 DDS_ANY_VIEW_STATE

```
const DDS_ViewStateMask DDS_ANY_VIEW_STATE [extern]
```

Any view state **DDS_NEW_VIEW_STATE** (p. 713) | **DDS_NOT_NEW_VIEW_STATE** (p. 713).

Examples

HelloWorld_subscriber.c.

5.28 Instance States

DDS_InstanceStateKind (p. 714) and associated elements

Typedefs

- typedef **DDS_UnsignedLong DDS_InstanceStateMask**
A bit-mask (list) of instance states, i.e. **DDS_InstanceStateKind** (p. 714).

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 **DDS_DataWriter** (p. 485) 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**.

5.28.1 Detailed Description

DDS_InstanceStateKind (p. 714) and associated elements

5.28.2 Typedef Documentation

5.28.2.1 DDS_InstanceStateMask

```
typedef DDS_UnsignedLong DDS_InstanceStateMask
```

A bit-mask (list) of instance states, i.e. **DDS_InstanceStateKind** (p. 714).

5.28.3 Enumeration Type Documentation

5.28.3.1 DDS_InstanceStateKind

```
enum DDS_InstanceStateKind
```

Indicates if the samples are from a live **DDS_DataWriter** (p. 485) or not.

For each instance, the middleware internally maintains an instance state. The instance state can be:

- **DDS_ALIVE_INSTANCE_STATE** (p. 715) indicates that (a) samples have been received for the instance, (b) there are live **DDS_DataWriter** (p. 485) 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. 715) indicates the instance was explicitly disposed by a **DDS_DataWriter** (p. 485) by means of the dispose operation.
- **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 715) indicates the instance has been declared as not-alive by the **DDS_DataReader** (p. 615) because it detected that there are no live **DDS_DataWriter** (p. 485) 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. 1301) is set to **DDS_EXCLUSIVE_OWNERSHIP_QOS** (p. 1302), then the instance state becomes **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 715) only if the **DDS_DataWriter** (p. 485) that "owns" the instance explicitly disposes it. The instance state becomes **DDS_ALIVE_INSTANCE_STATE** (p. 715) again only if the **DDS_DataWriter** (p. 485) that owns the instance writes it.
- If **OWNERSHIP** (p. 1301) is set to **DDS_SHARED_OWNERSHIP_QOS** (p. 1302), then the instance state becomes **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 715) if any **DDS_DataWriter** (p. 485) explicitly disposes the instance. The instance state becomes **DDS_ALIVE_INSTANCE_STATE** (p. 715) as soon as any **DDS_DataWriter** (p. 485) writes the instance again.

The instance state available in the **DDS_SampleInfo** (p. 1985) 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 DDS_DataWriter (p. 485) objects that are currently alive (according to the LIVELINESS (p. 1296)) are writing the instance.

5.28.4 Variable Documentation

5.28.4.1 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.c.

5.28.4.2 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.

5.29 Stream Kinds

DDS_StreamKind (p. 716) and associated elements

Typedefs

- typedef **DDS_UnsignedLong** **DDS_StreamKindMask**
*A bit-mask (list) of stream kinds, i.e. **DDS_StreamKind** (p. 716).*

Enumerations

- enum **DDS_StreamKind** {
DDS_LIVE_STREAM = 0x0001 << 0 ,
DDS_TOPIC_QUERY_STREAM = 0x0001 << 1 }
Indicates if the samples are TopicQuery samples or not.

5.29.1 Detailed Description

DDS_StreamKind (p. 716) and associated elements

5.29.2 Typedef Documentation

5.29.2.1 DDS_StreamKindMask

```
typedef DDS_UnsignedLong DDS_StreamKindMask
```

A bit-mask (list) of stream kinds, i.e. **DDS_StreamKind** (p. 716).

5.29.3 Enumeration Type Documentation

5.29.3.1 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.

5.30 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. 1012) GUID type and defines.
- **Sequence Number Support**
<<extension>> (p. 1012) Sequence number type and defines.
- **Exception Codes**
<<extension>> (p. 1012) Exception codes.
- **Return Codes**
Types of return codes.
- **Status Kinds**
Kinds of communication status.
- **QoS**
Quality of Service (QoS) policies.
- **Entity Support**
***DDS_Entity** (p. 1360), **DDS_Listener** (p. 1816) and related items.*
- **Conditions and WaitSets**
***DDS_Condition** (p. 1370) and **DDS_WaitSet** (p. 1371) and related items.*
- **Cookie**
<<extension>> (p. 1012) Unique identifier for a written data sample
- **Sample Flags**
<<extension>> (p. 1012) Flags for samples.
- **WriteParams**
<<extension>> (p. 1012)
- **Heap Support in C**
<<extension>> (p. 1012) Heap allocation and free routines in C
- **Builtin Qos Profiles**
<<extension>> (p. 1012) QoS libraries, profiles, and snippets that are automatically built into RTI Connex.
- **User-managed Threads**
User-managed thread infrastructure.
- **Octet Buffer Support**
<<extension>> (p. 1012) Octet buffer creation, cloning, and deletion.
- **Sequence Support**
*The **FooSeq** (p. 2113) interface allows you to work with variable-length collections of homogeneous data.*
- **String Support**
<<extension>> (p. 1012) String creation, cloning, assignment, and deletion.

5.30.1 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"

5.31 Built-in Sequences

Defines sequences of primitive data type. .

Data Structures

- struct **DDS_CharSeq**
*Instantiates **FooSeq** (p. 2113) < **DDS_Char** (p. 1201) >*
- struct **DDS_WcharSeq**
*Instantiates **FooSeq** (p. 2113) < **DDS_Wchar** (p. 1202) >*
- struct **DDS_OctetSeq**
*Instantiates **FooSeq** (p. 2113) < **DDS_Octet** (p. 1202) >*
- struct **DDS_UInt8Seq**
*Instantiates **FooSeq** (p. 2113) < **DDS_UInt8** (p. 1202) >*
- struct **DDS_Int8Seq**
*Instantiates **FooSeq** (p. 2113) < **DDS_Int8** (p. 1202) >*
- struct **DDS_ShortSeq**
*Instantiates **FooSeq** (p. 2113) < **DDS_Short** (p. 1202) >*
- struct **DDS_UnsignedShortSeq**
*Instantiates **FooSeq** (p. 2113) < **DDS_UnsignedShort** (p. 1202) >*
- struct **DDS_LongSeq**
*Instantiates **FooSeq** (p. 2113) < **DDS_Long** (p. 1203) >*
- struct **DDS_UnsignedLongSeq**
*Instantiates **FooSeq** (p. 2113) < **DDS_UnsignedLong** (p. 1203) >*
- struct **DDS_LongLongSeq**
*Instantiates **FooSeq** (p. 2113) < **DDS_LongLong** (p. 1203) >*
- struct **DDS_UnsignedLongLongSeq**
*Instantiates **FooSeq** (p. 2113) < **DDS_UnsignedLongLong** (p. 1203) >*
- struct **DDS_FloatSeq**
*Instantiates **FooSeq** (p. 2113) < **DDS_Float** (p. 1203) >*
- struct **DDS_DoubleSeq**
*Instantiates **FooSeq** (p. 2113) < **DDS_Double** (p. 1203) >*
- struct **DDS_LongDoubleSeq**
*Instantiates **FooSeq** (p. 2113) < **DDS_LongDouble** (p. 1204) >*
- struct **DDS_BooleanSeq**
*Instantiates **FooSeq** (p. 2113) < **DDS_Boolean** (p. 1204) >*
- struct **DDS_StringSeq**
*Instantiates **FooSeq** (p. 2113) < **char*** > with value type semantics.*
- struct **DDS_WstringSeq**
*Instantiates **FooSeq** (p. 2113) < **DDS_Wchar** (p. 1202)* >*

5.31.1 Detailed Description

Defines sequences of primitive data type. .

5.32 Multi-channel DataWriters

APIs related to Multi-channel DataWriters.

APIs related to Multi-channel DataWriters.

5.32.1 What is a Multi-channel DataWriter?

A Multi-channel **DDS_DataWriter** (p. 485) is a **DDS_DataWriter** (p. 485) 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 **DDS_DataWriter** (p. 485). Each filter "guards" a channel (a set of multicast addresses). Each time a multi-channel **DDS_DataWriter** (p. 485) 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.

5.32.2 Configuration on the Writer Side

To configure a multi-channel **DDS_DataWriter** (p. 485), simply define a list of all its channels in the **DDS_MultiChannelQosPolicy** (p. 1856).

The **DDS_MultiChannelQosPolicy** (p. 1856) is propagated along with discovery traffic. The value of this policy is available in **DDS_PublicationBuiltinTopicData::locator_filter** (p. 1912).

5.32.3 Configuration on the Reader Side

No special changes are required in a subscribing application to get data from a multichannel **DDS_DataWriter** (p. 485). If you want the **DDS_DataReader** (p. 615) to subscribe to only a subset of the channels, use a **DDS_ContentFilteredTopic** (p. 177).

For more information on Multi-channel DataWriters, refer to the `User's Manual`.

5.32.4 Reliability with Multi-Channel DataWriters

5.32.4.1 Reliable Delivery

Reliable delivery is only guaranteed when the **DDS_PresentationQosPolicy::access_scope** (p. 1893) is set to **DDS_INSTANCE_PRESENTATION_QOS** (p. 1305) and the filters in **DDS_MultiChannelQosPolicy** (p. 1856) 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.

5.32.4.2 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. 1951) and/or **DDS_DataWriterResourceLimitsQosPolicy::max_batches** (p. 1677).

The protocol parameters described in **DDS_DataWriterProtocolQosPolicy** (p. 1649) are applied per channel, with the following exceptions:

DDS_RtpsReliableWriterProtocol_t::low_watermark (p. 1961) and **DDS_RtpsReliableWriterProtocol_t::high_watermark** (p. 1962): 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. 1967): 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_ResourceLimitsQosPolicy::max_samples** (p. 1951))

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. 1951) instead of **max_batches** **DDS_DataWriterResourceLimitsQosPolicy::max_batches** (p. 1677) in order to take advantage of **heartbeats_per_max_samples**.

5.33 Transports

APIs related to RTI Connex transportable transports.

Modules

- **Installing Transport Plugins**

Installing and configuring transports used by RTI Connex.

- **Built-in Transport Plugins**

Transport plugins delivered with RTI Connex.

- **Creating New Transport Plugins**

Developing new transport plugins for RTI Connex.

- **Transport Plugins Configuration**

Transport plugins configuration with RTI Connex.

- **Transport Address**

Transport-independent addressing scheme using IPv6 presentation strings and numerically stored in network-ordered format.

5.33.1 Detailed Description

APIs related to RTI Connext pluggable transports.

5.33.2 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. 736)) 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. 2051)). 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.

5.33.3 Transport Aliases

In order to use a transport plugin instance in an RTI Connext application, it must be registered with a **DDS_DomainParticipant** (p. 77). 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 **DDS_DomainParticipant** (p. 77). An **alias** symbolically refers to one or more transport plugins registered with the **DDS_DomainParticipant** (p. 77). Builtin transport plugin instances can be referred to using preconfigured aliases (see **TRANSPORT_BUILTIN** (p. 1331)).

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. 1716)), and for **DDS_DataWriter** (p. 485) and **DDS_DataReader** (p. 615) entities (see **DDS_TransportSelectionQosPolicy** (p. 2064)).
- the **multicast** addresses on which to receive discovery messages (see **DDS_DiscoveryQosPolicy::multicast_receive_addresses** (p. 1717)), and the multicast addresses and ports on which to receive user data (see **DDS_DataReaderQos::multicast** (p. 1616)).
- the **unicast ports** used for user data (see **DDS_TransportUnicastQosPolicy** (p. 2065)) on both **DDS_DataWriter** (p. 485) and **DDS_DataReader** (p. 615) entities.
- the transport plugins used to parse an address string in a locator (**Locator Format** (p. 1353) and **NDDS_DISCOVERY_PEERS** (p. 1352)).

A **DDS_DomainParticipant** (p. 77) (and contained its entities) start using a transport plugin after the **DDS_DomainParticipant** (p. 77) is enabled (see **DDS_Entity_enable** (p. 1363)). 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" than others.

5.33.4 Transport Lifecycle

A transport plugin is owned by whomever creates it. Thus, if you create and register a transport plugin with a **DDS_↔DomainParticipant** (p. 77), you are responsible for deleting it by calling its destructor. Note that builtin transport plugins (**TRANSPORT_BUILTIN** (p. 1331)) and transport plugins that are loaded through the **PROPERTY** (p. 1306) QoS policy (see **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 727)) are automatically managed by RTI Connex.

A user-created transport plugin must not be deleted while it is still in use by a **DDS_DomainParticipant** (p. 77). This generally means that a user-created transport plugin instance can only be deleted after the **DDS_DomainParticipant** (p. 77) with which it was registered is deleted (see **DDS_DomainParticipantFactory_delete_participant** (p. 44)). Note that a transport plugin *cannot* be "unregistered" from a **DDS_DomainParticipant** (p. 77).

A transport plugin instance cannot be registered with more than one **DDS_DomainParticipant** (p. 77) 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 **DDS_DomainParticipant** (p. 77) 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 **DDS_DomainParticipant** (p. 77) 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.

5.33.5 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. 2123)) 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.

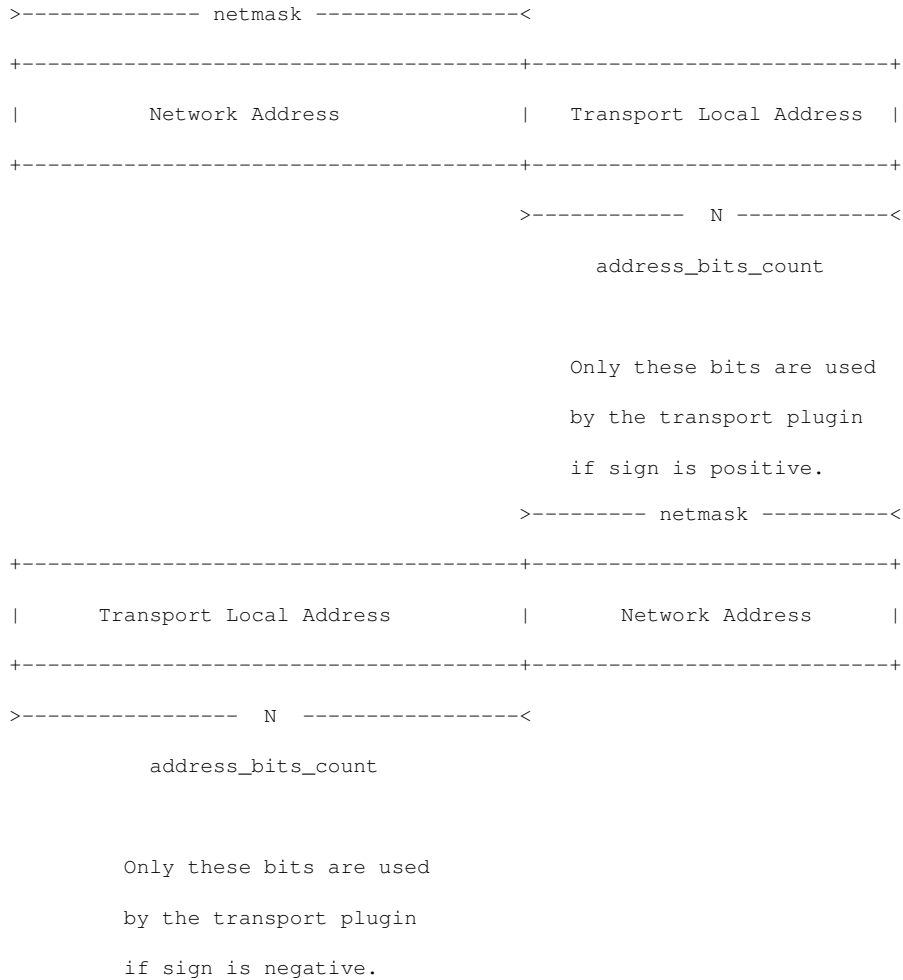
Two transport plugin instances report the same class *iff* they have compatible implementations. Transport plugin instances with mismatching classes are not allowed (by the RTI Connex Core) to communicate with one another.

Multiple implementations (possibly from different vendors) for a physical transport mechanism can co-exist in an RTI Connex 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. 2123)) RTI Connex's addressing is modeled after the IPv6 and uses 128-bit addresses (**Transport Address** (p. 1027)) 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 *N* least significant bits (LSB) if positive or *N* most significant bits (MSB) if negative; these bits are specified by this attribute.



The remaining bits of an address using the $128 - \text{abs}(\text{bit address})$ representation will be considered as part of the "network address" (see **Transport Network Address** (p. 724)) and thus ignored by the transport plugin's internal addressing scheme.

For *unicast* addresses, the transport plugin is expected to ignore the higher $(128 - \text{NDDS\_Transport\_Property\_} \leftarrow \text{t::address\_bit\_count})$ bits. RTI Connex 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:

$$\text{address\_bits\_count} \geq \text{ceil}(\log_2(\text{total\_addressable\_transport\_unicast\_interfaces}))$$

The equality holds when the most compact (theoretical) internal address mapping scheme is used. A practical address mapping scheme may waste some bits.

5.33.6 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. 2124) and **NDDS_Transport_Property_t::gather_send_buffer_count_max** (p. 2124).

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.

5.33.7 Transport Network Address

The address bits not used by the transport plugin for its internal addressing constitute its network address bits.

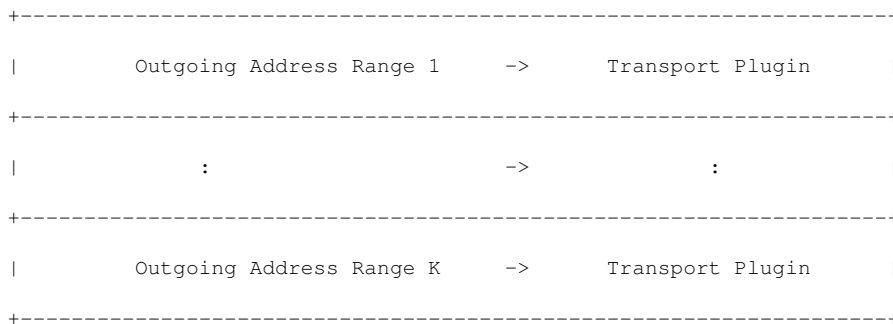
In order for RTI Connext to properly route the messages, each unicast interface in the RTI Connext *domain* must have a unique address. RTI Connext allows the user to specify the value of the network address when installing a transport plugin via the **NDDS_Transport_Support_register_transport()** (p. 730) 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 **DDS_DomainParticipant** (p. 77), 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).

5.33.8 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 **NDDS_Transport_Support_add_send_route()** (p. 732) API, so that a transport plugin will be used to send messages only to certain ranges of destination addresses. The function 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.

5.33.9 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 **NDDS_Transport_Support_add_receive_route()** (p. 733) API, so that a transport plugin will be used to receive messages only on certain ranges of addresses. The function can be called multiple times for a transport plugin, with different address ranges.

```
+-----+
|          Transport Plugin          <- Incoming Address Range 1 |
+-----+
|          :          <-          :          |
+-----+
|          Transport Plugin          <- Incoming Address Range M |
+-----+
```

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).

5.34 Installing Transport Plugins

Installing and configuring transports used by RTI Connext.

Data Structures

- struct **NDDS_Transport_Support**
 <<*interface*>> (p. 1013) *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 **DDS_DomainParticipant** (p. 77).*
- 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. 1901).*

Functions

- **DDS_Boolean NDDS_Transport_Handle_is_nil** (const **NDDS_Transport_Handle_t** *self)
Is the given transport handle the NIL transport handle?
- * **NDDS_Transport_Handle_t NDDS_Transport_Support_register_transport** (**DDS_DomainParticipant** *participant_in, **NDDS_Transport_Plugin** *transport_in, const struct **DDS_StringSeq** *aliases_in, const **NDDS_Transport_Address_t** *network_address_in)
*Register a transport plugin for use with a **DDS_DomainParticipant** (p. 77), assigning it a network_address.*
- **NDDS_Transport_Handle_t NDDS_Transport_Support_lookup_transport** (**DDS_DomainParticipant** *participant_in, struct **DDS_StringSeq** *aliases_out, **NDDS_Transport_Address_t** *network_address_out, const **NDDS_Transport_Plugin** *transport_in)
*Look up a transport plugin within a **DDS_DomainParticipant** (p. 77).*
- **DDS_ReturnCode_t NDDS_Transport_Support_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.
- **DDS_ReturnCode_t NDDS_Transport_Support_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.
- **DDS_ReturnCode_t NDDS_Transport_Support_get_builtin_transport_property** (**DDS_DomainParticipant** *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.
- **DDS_ReturnCode_t NDDS_Transport_Support_set_builtin_transport_property** (**DDS_DomainParticipant** *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.
- **NDDS_Transport_Plugin * NDDS_Transport_Support_get_transport_plugin** (**DDS_DomainParticipant** *participant_in, const char *alias_in)
*Retrieve a transport plugin registered in a **DDS_DomainParticipant** (p. 77) by its alias.*

Variables

- const **NDDS_Transport_Handle_t NDDS_TRANSPORT_HANDLE_NIL**
The NIL transport handle.

5.34.1 Detailed Description

Installing and configuring transports used by RTI Connex.

There is more than one way to install a transport plugin for use with RTI Connex:

- If it is a builtin transport plugin, by specifying a bitmask in **DDS_TransportBuiltinQosPolicy** (p. 2051) (see **Builtin Transport Plugins** (p. 736))

- For all other non-builtin transport plugins, by dynamically loading the plugin through **PROPERTY** (p. 1306) QoS policy settings of **DDS_DomainParticipant** (p. 77) (only supported on architectures that support dynamic libraries, see **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 727))
- By explicitly creating a transport plugin and registering the plugin with a **DDS_DomainParticipant** (p. 77) through **NDDS_Transport_Support_register_transport** (p. 730) (for both builtin and non-builtin plugins)

In the first two cases, the lifecycle of the transport plugin is automatically managed by RTI Connex. In the last case, user is responsible for deleting the transport plugin after the **DDS_DomainParticipant** (p. 77) is deleted. See **Transport Lifecycle** (p. 722) for details.

5.34.2 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 Connex through the **PROPERTY** (p. 1306) QoS policy settings of the **DDS_DomainParticipant** (p. 77).

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 Connex abstract transport API and must provide a function with the signature **NDDS_Transport_create_plugin** (p. 729) that can be called by RTI Connex 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. 1306) QoS policy of the **DDS_DomainParticipant** (p. 77).

The following table lists the property names that are used to load the transport plugins dynamically:

Table 5.711 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 Connex. 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

Property Name	Description	Required?
<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. 729) 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 NDDS_Transport_Support_register_transport (p. 730)	YES
<TRANSPORT_PREFIX>.aliases	Used to register the transport plugin returned by NDDS_Transport_create_plugin (p. 729) (as specified by <TRANSPORT_PREFIX>.create_function) to the DDS_DomainParticipant (p. 77). Refer to aliases_in parameter in NDDS_Transport_Support_register_transport (p. 730) 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
<TRANSPORT_PREFIX>.network_address	Used to register the transport plugin returned by NDDS_Transport_create_plugin (p. 729) (as specified by <TRANSPORT_PREFIX>.create_function) to the DDS_DomainParticipant (p. 77). Refer to network_address_in parameter in NDDS_Transport_Support_register_transport (p. 730) for details. If it is not specified, the network_address_out output parameter from NDDS_Transport_create_plugin (p. 729) is used. The default value is a zeroed out network address.	NO

Property Name	Description	Required?
<TRANSPORT_PREFIX>.<property_name>	Property that is passed into NDDS_Transport_create_plugin (p. 729) (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. 729) 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 **DDS_DomainParticipant** (p. 77) by RTI Connnext when:

- the **DDS_DomainParticipant** (p. 77) is enabled,
- the first DataWriter/DataReader is created, or
- you lookup a builtin DataReader (**DDS_Subscriber_lookup_datareader** (p. 586)),

whichever happens first.

Any changes to the transport plugin related properties in the **PROPERTY** (p. 1306) QoS policy after the transport plugin has been registered with the **DDS_DomainParticipant** (p. 77) will have no effect.

See also

Transport Use Cases (p. 992)

5.34.3 Typedef Documentation

5.34.3.1 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 **DDS_DomainParticipant** (p. 77).

A transport handle represents the association between a **DDS_DomainParticipant** (p. 77) and a transport plugin.

5.34.3.2 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. 1901).

By specifying some predefined property names in **DDS_PropertyQosPolicy** (p. 1901), RTI Connexx can call a function from a dynamic library to create a transport plugin and register the returned plugin with a **DDS_DomainParticipant** (p. 77).

This is the function prototype of the function as specified in "<TRANSPORT_PREFIX>.create_function" of **DDS_PropertyQosPolicy** (p. 1901) QoS policy that will be called by RTI Connexx to create the transport plugin. See **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 727) for details.

Parameters

<i>default_network_address_out</i>	<< <i>out</i> >> (p. 1013) Optional output parameter. If the network address is not specified in "<TRANSPORT_PREFIX>.network_address" in DDS_PropertyQosPolicy (p. 1901), this is the default network address that is used to register the returned transport plugin using NDDS_Transport_Support_register_transport (p. 730) . This parameter will never be null. The default value is a zeroed-out network address.
<i>property_in</i>	<< <i>in</i> >> (p. 1013) property_in contains all the name-value pair properties that matches the format "<TRANSPORT_PREFIX>.<property_name>" in DDS_PropertyQosPolicy (p. 1901) 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.

5.34.4 Function Documentation

5.34.4.1 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. 1201) if the given transport handle is equal to **NDDS_TRANSPORT_HANDLE_NIL** (p. 736) or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.34.4.2 NDDS_Transport_Support_register_transport()

```
* NDDS_Transport_Handle_t NDDS_Transport_Support_register_transport (
    DDS_DomainParticipant * participant_in,
    NDDS_Transport_Plugin * transport_in,
    const struct DDS_StringSeq * aliases_in,
    const NDDS_Transport_Address_t * network_address_in )
```

Register a transport plugin for use with a **DDS_DomainParticipant** (p. 77), assigning it a `network_address`.

A transport plugin instance can be used by exactly one **DDS_DomainParticipant** (p. 77) at a time.

When a DataWriter/DataReader is created, only those transports already registered to the corresponding **DDS_DomainParticipant** (p. 77) 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. 736) for details on how to control the builtin transports that are automatically registered.

Precondition

A disabled **DDS_DomainParticipant** (p. 77) and a transport plugin that will be registered exclusively with it.

Parameters

<i>participant_in</i>	<< <i>in</i> >> (p. 1013) A non-null, disabled DDS_DomainParticipant (p. 77).
<i>transport_in</i>	<< <i>in</i> >> (p. 1013) A non-null transport plugin that is currently not registered with another DDS_DomainParticipant (p. 77).
<i>aliases_in</i>	<< <i>in</i> >> (p. 1013) 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 DDS_Entity (p. 1360) in the DDS_DomainParticipant (p. 77) if the transport alias list associated with the DDS_Entity (p. 1360) 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. 1331).
<i>network_address_in</i>	<< <i>in</i> >> (p. 1013) The network address at which to register this transport plugin. The bits indicated by <code>transport_in.property.address_bit_count</code> will be truncated. The remaining bits are the network address of the transport plugin. (see Transport Class Attributes (p. 722)).

Returns

Upon success, a valid non-NIL transport handle, representing the association between the **DDS_DomainParticipant** (p. 77) and the transport plugin; a **NDDS_TRANSPORT_HANDLE_NIL** (p. 736) 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. 722)

Transport Network Address (p. 724)

Locator Format (p. 1353)

NDDS_DISCOVERY_PEERS (p. 1352)

5.34.4.3 NDDS_Transport_Support_lookup_transport()

```
NDDS_Transport_Handle_t NDDS_Transport_Support_lookup_transport (
    DDS_DomainParticipant * participant_in,
    struct DDS_StringSeq * aliases_out,
    NDDS_Transport_Address_t * network_address_out,
    const NDDS_Transport_Plugin * transport_in )
```

Look up a transport plugin within a **DDS_DomainParticipant** (p. 77).

The transport plugin should have already been registered with the **DDS_DomainParticipant** (p. 77).

Parameters

<i>participant_in</i>	<< <i>in</i> >> (p. 1013) A non-null DDS_DomainParticipant (p. 77).
<i>aliases_out</i>	<< <i>inout</i> >> (p. 1013) A sequence of string where the aliases used to refer to the transport plugin symbolically will be returned. null if not interested.
<i>network_address_out</i>	<< <i>inout</i> >> (p. 1013) The network address at which to register the transport plugin will be returned here. null if not interested.
<i>transport_in</i>	<< <i>in</i> >> (p. 1013) A non-null transport plugin that is already registered with the DDS_DomainParticipant (p. 77).

Returns

Upon success, a valid non-NIL transport handle, representating the association between the **DDS_DomainParticipant** (p. 77) and the transport plugin; a **NDDS_TRANSPORT_HANDLE_NIL** (p. 736) upon failure.

See also

Transport Class Attributes (p. 722)

Transport Network Address (p. 724)

5.34.4.4 NDDS_Transport_Support_add_send_route()

```
DDS_ReturnCode_t NDDS_Transport_Support_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.

This function can be used to narrow the range of addresses to which outgoing messages can be sent.

Precondition

A disabled **DDS_DomainParticipant** (p. 77).

Parameters

<i>transport_handle_in</i>	<< <i>in</i> >> (p. 1013) A valid non-NIL transport handle as a result of a call to NDDS_Transport_Support_register_transport() (p. 730).
<i>address_range_in</i>	<< <i>in</i> >> (p. 1013) The outgoing address range for which to use this transport plugin.
<i>address_range_bit_count_in</i>	<< <i>in</i> >> (p. 1013) 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. 2123).

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

See also

Transport Send Route (p. 724)

5.34.4.5 NDDS_Transport_Support_add_receive_route()

```
DDS_ReturnCode_t NDDS_Transport_Support_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.

This function can be used to narrow the range of addresses at which to receive incoming messages.

Precondition

A disabled **DDS_DomainParticipant** (p. 77).

Parameters

<i>transport_handle_in</i>	<< <i>in</i> >> (p. 1013) A valid non-NIL transport handle as a result of a call to NDDS_Transport_Support_register_transport() (p. 730).
<i>address_range_in</i>	<< <i>in</i> >> (p. 1013) The incoming address range for which to use this transport plugin.
<i>address_range_bit_count_in</i>	<< <i>in</i> >> (p. 1013) 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. 2123).

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

See also

Transport Receive Route (p. 725)

5.34.4.6 NDDS_Transport_Support_get_builtin_transport_property()

```

DDS_ReturnCode_t NDDS_Transport_Support_get_builtin_transport_property (
    DDS_DomainParticipant * 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.

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

<i>participant_in</i>	<< <i>in</i> >> (p. 1013) A valid non-null DDS_DomainParticipant (p. 77)
<i>builtin_transport_kind_in</i>	<< <i>in</i> >> (p. 1013) The builtin transport kind for which to retrieve the properties.
<i>builtin_transport_property_inout</i>	<< <i>inout</i> >> (p. 1013) The storage area where the retrieved property will be output. The specific type required by the <i>builtin_transport_kind_in</i> must be used.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

See also

NDDS_Transport_Support_set_builtin_transport_property() (p. 735)

5.34.4.7 NDDS_Transport_Support_set_builtin_transport_property()

```
DDS_ReturnCode_t NDDS_Transport_Support_set_builtin_transport_property (
    DDS_DomainParticipant * 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.

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. 736) for details on when a builtin transport is registered.

Precondition

A disabled **DDS_DomainParticipant** (p. 77). The `builtin_transport_property_inout` parameter must be of the type specified by the `builtin_transport_kind_in`.

Parameters

<i>participant_in</i>	<< <i>in</i> >> (p. 1013) A valid non-null DDS_DomainParticipant (p. 77) that has not been enabled.
<i>builtin_transport_kind_in</i>	<< <i>in</i> >> (p. 1013) The builtin transport kind for which to specify the properties.
<i>builtin_transport_property_in</i>	<< <i>inout</i> >> (p. 1013) The new transport property that will be used to create the builtin transport plugin. The specific type required by the <code>builtin_transport_kind_in</code> must be used.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

See also

NDDS_Transport_Support_get_builtin_transport_property() (p. 734)

5.34.4.8 NDDS_Transport_Support_get_transport_plugin()

```
NDDS_Transport_Plugin * NDDS_Transport_Support_get_transport_plugin (
    DDS_DomainParticipant * participant_in,
    const char * alias_in )
```

Retrieve a transport plugin registered in a **DDS_DomainParticipant** (p. 77) by its alias.

This method can be used to get a pointer to a transport Plugin that has been registered into the **DDS_DomainParticipant** (p. 77).

Parameters

<i>participant_in</i>	<<in>> (p. 1013) A non-null DDS_DomainParticipant (p. 77).
<i>alias_in</i>	<<in>> (p. 1013) 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.

5.34.5 Variable Documentation

5.34.5.1 NDDS_TRANSPORT_HANDLE_NIL

```
const NDDS_Transport_Handle_t NDDS_TRANSPORT_HANDLE_NIL [extern]
```

The NIL transport handle.

5.35 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. 1024)).*

- **UDPv4 Transport**

*Transport plug-in using UDP/IPv4 (**NDDS_TRANSPORT_CLASSID_UDPv4** (p. 1023)).*

- **Real-Time WAN Transport**

*Transport plug-in using UDP/IPv4 for WAN communications. (**NDDS_TRANSPORT_CLASSID_UDPv4_WAN** (p. 1025)).*

- **UDPv6 Transport**

*Transport plug-in using UDP/IPv6 (**NDDS_TRANSPORT_CLASSID_UDPv6** (p. 1024)).*

5.35.1 Detailed Description

Transport plugins delivered with RTI Connext.

The **TRANSPORT_BUILTIN** (p. 1331) 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 specifying a bitmask in **DDS_TransportBuiltinQosPolicy** (p. 2051), 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. 2051), the plugin is implicitly created and registered to the corresponding **DDS_DomainParticipant** (p. 77) by RTI Connext when:

- the **DDS_DomainParticipant** (p. 77) is enabled,
- the first DataWriter/DataReader is created, or
- you lookup a builtin DataReader (**DDS_Subscriber_lookup_datareader** (p. 586)),

whichever happens first.

Each builtin transport contains its own set of properties. For example, the **::UDpv4 Transport** (p. 1041) 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 function **NDDS_Transport_Support_set_builtin_transport_property()** (p. 735) or by using the **PROPERTY** (p. 1306) QoS policy associated with the **DDS_DomainParticipant** (p. 77). Builtin transport plugin properties specified in **DDS_PropertyQosPolicy** (p. 1901) always overwrite the ones specified through **NDDS_Transport_Support_set_builtin_transport_property()** (p. 735). Refer to the specific builtin transport for the list of property names that can be specified through **PROPERTY** (p. 1306) QoS policy.

Any changes to the builtin transport properties after the builtin transports have been registered with will have no effect.

See also

NDDS_Transport_Support_set_builtin_transport_property() (p. 735) **DDS_PropertyQosPolicy** (p. 1901)

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. 994). This is useful when the application needs different values for the network addresses.

5.36 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.

5.37 Queries and Filters Syntax

5.37.1 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 **DDS_ContentFilteredTopic** (p. 177)
- The `query_expression` in the **DDS_QueryCondition** (p. 699)
- <<*extension*>> (p. 1012) The `filter_expression` in the **DDS_TopicQuerySelection** (p. 2049)
- <<*extension*>> (p. 1012) The `filter_expression` in the **DDS_ChannelSettings_t** (p. 1564)

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*.

5.37.2 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

```

5.37.3 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

- **TOPICNAME** - A topic name is an identifier for a topic, and is defined as any series of characters 'a', ..., 'z', 'A', ..., 'Z', '0', ..., '9', '_' but may not start with a digit.

Formal notation:

TOPICNAME : **IDENTIFIER**

- **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 *en* or *En*, 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* + 1th argument in the given context. Argument can only in primitive type value format. It cannot be a FIELDNAME.

Formal notation:

```
PARAMETER : "%" (["0" - "9"])+
```

5.37.4 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 "
```

5.37.5 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. 742)
- (*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 "MyEnum=MyString" is not allowed.
- (*3) Only for same type Enums.

5.37.6 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 <i>non-special</i> characters in the field string.
*	"*" in the pattern string matches 0 or more <i>non-special</i> characters in field string.
[<i>charlist</i>]	Matches any one of the characters from the list of characters in <i>charlist</i> .
[<i>s-e</i>]	Matches any character any character from <i>s</i> to <i>e</i> , inclusive.
%	"%" is used to designate filter expressions parameters.
[! <i>charlist</i>] or [^ <i>charlist</i>]	Matches any characters not in <i>charlist</i> (not supported).
[! <i>s-e</i>] or [^ <i>s-e</i>]	Matches any characters not in the interval [<i>s-e</i>] (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.

5.37.7 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**

5.37.8 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 `é` is represented by `\u00e9`
- Composed `é` = `e` + `´` is represented by `\u00e9` + `\u0027`

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.

5.37.9 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 **DDS_ContentFilteredTopic** (p. 177)) or a **query_expression** (for **DDS_QueryCondition** (p. 699)):

- `"z < 1000 AND x < 23"`

Examples of a **filter_expression** using **MATCH** (for **DDS_ContentFilteredTopic** (p. 177)) operator:

- `"symbol MATCH 'NASDAQ/GOOG' "`
- `"symbol MATCH 'NASDAQ/[A-M]*' "`

5.38 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.

5.38.1 Detailed Description

APIs of troubleshooting utilities and APIs designed to configure the overall behavior of RTI Connext.

5.39 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 **NDDS_Utility_spin** (p. 1499).*

5.39.1 Detailed Description

API of general utilities used in the RTI Connex distribution.

5.40 Observability Framework

RTI Connex Observability Framework modules and APIs.

Modules

- **Monitoring Library 2.0**

RTI Monitoring Library 2.0 public API.

5.40.1 Detailed Description

RTI Connex Observability Framework modules and APIs.

RTI Connex Observability Framework is a holistic solution that uses telemetry data to provide deep visibility into the current and past states of your RTI Connex applications.

5.41 Request-Reply Pattern

Support for the request-reply communication pattern.

Modules

- **Requester**

FooBarRequester (p. 2111) and associated elements

- **Replier**

FooBarReplier (p. 2110), *FooBarSimpleReplier* (p. 2112) and associated elements

5.41.1 Detailed Description

Support for the request-reply communication pattern.

There are two basic entities that enable this pattern:

- **FooBarRequester** (p. 2111)
- **FooBarReplier** (p. 2110) (and a simpler version **FooBarSimpleReplier** (p. 2112))

This functionality is built on top of RTI Connex.

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.

See also

Request-Reply Examples (p. 1002).

5.42 Requester

FooBarRequester (p.2111) and associated elements

Data Structures

- struct **RTI_Connext_Requester**
The type-independent version of a Requester.
- struct **RTI_Connext_RequesterParams**
*Contains the parameters for creating a **FooBarRequester** (p. 2111).*
- struct **FooBarRequester**
Allows sending requests and receiving replies.

Macros

- #define **RTI_CONNEXT_REQUESTER_DECL**(TReq, TRep, TRequester)
Instantiates the declaration of a typed requester and its operations.

Typedefs

- typedef struct **RTI_Connext_RequesterParams** **RTI_Connext_RequesterParams**
*Contains the parameters for creating a **FooBarRequester** (p. 2111).*
- typedef struct **FooBarRequester** **FooBarRequester**
Allows sending requests and receiving replies.

Functions

- **DDS_ReturnCode_t** **RTI_Connext_Requester_delete** (**RTI_Connext_Requester** *self)
Releases the internal entities created by this object.
- **DDS_ReturnCode_t** **RTI_Connext_Requester_wait_for_replies** (**RTI_Connext_Requester** *self, **DDS_Long** min_count, const struct **DDS_Duration_t** *max_wait)
Waits for replies to any request.
- **DDS_ReturnCode_t** **RTI_Connext_Requester_wait_for_replies_for_related_request** (**RTI_Connext_Requester** *self, **DDS_Long** min_count, const struct **DDS_Duration_t** *max_wait, const struct **DDS_SampleIdentity_t** *related_request_id)
Waits for replies to a specific request.
- **FooBarRequester** * **FooBarRequester_create** (**DDS_DomainParticipant** *participant, const char *service_name)
Creates a Requester with the minimum set of parameters.
- **FooBarRequester** * **FooBarRequester_create_w_params** (const **RTI_Connext_RequesterParams** *params)
Creates a Requester with parameters.
- **DDS_ReturnCode_t** **FooBarRequester_send_request** (**FooBarRequester** *self, const **Foo** *request)
Sends a request.

- **DDS_ReturnCode_t FooBarRequester_send_request_w_params** (FooBarRequester *self, const Foo *request, struct DDS_WriteParams_t *request_info)
Sends a request and gets back information about it that allows correlation with future replies.
- **DDS_ReturnCode_t FooBarRequester_receive_reply** (FooBarRequester *self, Bar *reply, struct DDS_↵ SampleInfo *sample_info, const struct DDS_Duration_t *max_wait)
Waits for a reply and copies its contents into a Sample.
- **DDS_ReturnCode_t FooBarRequester_receive_replies** (FooBarRequester *self, struct BarSeq *received_↵ _data, struct DDS_SampleInfoSeq *info_seq, DDS_Long min_count, DDS_Long max_count, const struct DDS_Duration_t *max_wait)
Waits for multiple replies and provides a loaned sequence to access them.
- **DDS_ReturnCode_t FooBarRequester_take_reply** (FooBarRequester *self, Bar *reply, struct DDS_↵ SampleInfo *sample_info)
Copies the contents of a reply into a Sample.
- **DDS_ReturnCode_t FooBarRequester_take_replies** (FooBarRequester *self, struct BarSeq *reply_seq, struct DDS_SampleInfoSeq *sample_info_seq, DDS_Long max_count)
Provides a loaned sequence to access the existing replies.
- **DDS_ReturnCode_t FooBarRequester_take_reply_for_related_request** (FooBarRequester *self, Bar *reply, struct DDS_SampleInfo *sample_info, const struct DDS_SampleIdentity_t *related_request_id)
Copies the contents of a reply for a specific request.
- **DDS_ReturnCode_t FooBarRequester_take_replies_for_related_request** (FooBarRequester *self, struct BarSeq *reply_seq, struct DDS_SampleInfoSeq *sample_info_seq, DDS_Long max_count, const struct DDS_SampleIdentity_t *related_request_id)
Provides a loaned sequence to access the existing replies for a specific request.
- **DDS_ReturnCode_t FooBarRequester_read_reply** (FooBarRequester *self, Bar *reply, struct DDS_↵ SampleInfo *sample_info)
Copies the contents of a reply into a Sample.
- **DDS_ReturnCode_t FooBarRequester_read_replies** (FooBarRequester *self, struct BarSeq *reply_seq, struct DDS_SampleInfoSeq *sample_info_seq, DDS_Long max_count)
Provides a loaned sequence to access the existing replies.
- **DDS_ReturnCode_t FooBarRequester_read_reply_for_related_request** (FooBarRequester *self, Bar *reply, struct DDS_SampleInfo *sample_info, const struct DDS_SampleIdentity_t *related_request_id)
Copies the contents of a reply for a specific request.
- **DDS_ReturnCode_t FooBarRequester_read_replies_for_related_request** (FooBarRequester *self, struct BarSeq *reply_seq, struct DDS_SampleInfoSeq *sample_info_seq, DDS_Long max_count, const struct DDS_SampleIdentity_t *related_request_id)
Provides a loaned sequence to access the existing replies for a specific request.
- **FooDataWriter * FooBarRequester_get_request_datawriter** (FooBarRequester *self)
Retrieves the underlying DDS_DataWriter (p. 485).
- **BarDataReader * FooBarRequester_get_reply_datareader** (FooBarRequester *self)
Retrieves the underlying DDS_DataReader (p. 615).
- **DDS_ReturnCode_t FooBarRequester_return_loan** (FooBarRequester *self, struct BarSeq *received_data, struct DDS_SampleInfoSeq *info_seq)
Returns samples previously received from the middleware.

5.42.1 Detailed Description

FooBarRequester (p. 2111) and associated elements

5.42.2 Macro Definition Documentation

5.42.2.1 RTI_CONNEXT_REQUESTER_DECL

```
#define RTI_CONNEXT_REQUESTER_DECL(
    TReq,
    TRep,
    TRequester )
```

Instantiates the declaration of a typed requester and its operations.

Parameters

<i>TReq</i>	The request type name
<i>TRep</i>	The reply type name
<i>TRequester</i>	The type name for the typed requester and the prefix for all the operations.

See also

Creating a Requester (p. 1004)

5.42.3 Typedef Documentation

5.42.3.1 RTI_Connext_RequesterParams

```
typedef struct RTI_Connext_RequesterParams RTI_Connext_RequesterParams
```

Contains the parameters for creating a **FooBarRequester** (p. 2111).

The following parameters are required to create a Requester:

- A **DDS_DomainParticipant** (p. 77) (**RTI_Connext_RequesterParams::participant** (p. 2174)), and
- Either a service name (**RTI_Connext_RequesterParams::service_name** (p. 2174))
- Or custom topic names (**RTI_Connext_RequesterParams::request_topic_name** (p. 2174) and **RTI_Connext_RequesterParams::reply_topic_name** (p. 2175))

The rest of the parameters that can be set in a RequesterParams object are optional.

See also

Creating a Requester with optional parameters (p. 1004)

5.42.3.2 FooBarRequester

```
typedef struct FooBarRequester FooBarRequester
```

Allows sending requests and receiving replies.

A requester is an entity with two associated **topics** (p. 167): 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. 305), and **DDS_DynamicData** (p. 1768). **Note:** At this moment, in the C version of this API, only rtiddsgen-generated types are supported.

To create a Requester for two types, a request type TReq=Foo (p. 2109) and a reply type TRep=Bar, your application needs to instantiate the data structure **FooBarRequester** (p. 2111) and the specific operations that can publish and subscribe to those types. In this documentation we refer to the type-dependent operations as **FooBarRequester_** (for example, **FooBarRequester_take_replies** (p. 756)). Some operations are type-independent and their name always begins with **RTI_Connext_Requester_** (for example, **RTI_Connext_Requester_wait_for_replies** (p. 750)).

See **Creating a Requester** (p. 1004) to see how to instantiate a **FooBarRequester** (p. 2111).

A Replier and a Requester communicate when they use the same topics for requests and replies (see **RTI_Connext_RequesterParams::service_name** (p. 2174)) on the same **domain** (p. 77).

A Requester 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 receives multiple replies from a Replier for a given request, the Requester 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. 1388) is set in **DDS_SampleInfo::flag** (p. 1994) after receiving each reply. This indicates it is NOT the last reply.

A requester has an associated **DDS_DomainParticipant** (p. 77), 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 **DDS_DataWriter** (p. 485) for writing requests and a **DDS_DataReader** (p. 615) for reading replies, are automatically created when the requester is constructed.

Quality of Service for the underlying DataWriter and DataReader can be configured (see **RTI_Connext_RequesterParams::qos_profile_name** (p. 2175)). By default, they are created with **DDS_RELIABLE_RELIABILITY_QOS** (p. 1323). The exact default configuration is described here: **Configuring Request-Reply QoS profiles** (p. 1010)

See also

FooBarReplier (p. 2110)

Request-Reply Examples (p. 1002)

Requester example (p. 1005)

5.42.4 Function Documentation

5.42.4.1 RTI_Connext_Requester_delete()

```
DDS_ReturnCode_t RTI_Connext_Requester_delete (
    RTI_Connext_Requester * self )
```

Releases the internal entities created by this object.

Among other internal resources, it deletes the underlying DataReader and DataWriter.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

See also

DDS_Subscriber_delete_datareader (p. 585)

DDS_Publisher_delete_datawriter (p. 456)

5.42.4.2 RTI_Connext_Requester_wait_for_replies()

```
DDS_ReturnCode_t RTI_Connext_Requester_wait_for_replies (
    RTI_Connext_Requester * self,
    DDS_Long min_count,
    const struct 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 **FooBarRequester_take_replies** (p. 756)), 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. 1805) used to configure this Requester.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>min_count</i>	Minimum number of replies that need to be available for this operation to unblock.
<i>max_wait</i>	Maximum waiting time after which this operation unblocks, regardless of how many replies are available.

If at least `min_count` replies were available before `max_wait` elapsed, it returns **DDS_RETCODE_OK** (p. 1222), otherwise it returns **DDS_RETCODE_TIMEOUT** (p. 1222). Another **Standard Return Codes** (p. 1221) may be returned in case of error.

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 **RTI_Connext_Requester_wait_for_replies_for_related_request** (p. 751)).

See also

FooBarRequester_take_replies (p. 756)

5.42.4.3 RTI_Connext_Requester_wait_for_replies_for_related_request()

```
DDS_ReturnCode_t RTI_Connext_Requester_wait_for_replies_for_related_request (
    RTI_Connext_Requester * self,
    DDS_Long min_count,
    const struct DDS_Duration_t * max_wait,
    const struct DDS_SampleIdentity_t * related_request_id )
```

Waits for replies to a specific request.

This operation is analogous to **RTI_Connext_Requester_wait_for_replies** (p. 750) except this operation waits for replies for a specific request.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>min_count</i>	Minimum number of replies for the related request that need to be available for this operation to unblock.
<i>max_wait</i>	Maximum wait time after which this operation unblocks, regardless of how many replies are available.
<i>related_request_id</i>	The identity of a request previously sent by this Requester

Returns

true if at least `min_count` replies for the related request were available before `max_wait` elapsed, or false otherwise.
One of the **Standard Return Codes** (p. 1221)

MT Safety:

SAFE

See also

RTI_Connext_Requester_wait_for_replies (p. 750)

5.42.4.4 FooBarRequester_create()

```
FooBarRequester * FooBarRequester_create (
    DDS_DomainParticipant * participant,
    const char * service_name )
```

Creates a Requester with the minimum set of parameters.

Parameters

<i>participant</i>	The DomainParticipant this requester uses to join a DDS domain
<i>service_name</i>	The service name. See RTI_Connext_RequesterParams::service_name (p. 2174)

Returns

One of the **Standard Return Codes** (p. 1221)

5.42.4.5 FooBarRequester_create_w_params()

```
FooBarRequester * FooBarRequester_create_w_params (
    const RTI_Connext_RequesterParams * params )
```

Creates a Requester with parameters.

Parameters

<i>params</i>	All the parameters that configure this requester. See RTI_Connext_RequesterParams (p. 2173) for the list of mandatory parameters.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

RTI_Connext_RequesterParams (p. 2173)

Creating a Requester (p. 1004)

5.42.4.6 FooBarRequester_send_request()

```
DDS_ReturnCode_t FooBarRequester_send_request (
    FooBarRequester * self,
    const Foo * request )
```

Sends a request.

If a future reply needs to be correlated to exactly this request, use **FooBarRequester_send_request_w_params** (p. 753).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>request</i>	The request to be sent

Returns

One of the **Standard Return Codes** (p. 1221)

MT Safety:

SAFE

See also

FooBarRequester_send_request_w_params (p. 753)

5.42.4.7 FooBarRequester_send_request_w_params()

```
DDS_ReturnCode_t FooBarRequester_send_request_w_params (
    FooBarRequester * self,
    const Foo * request,
    struct DDS_WriteParams_t * request_info )
```

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. 1006).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>request</i>	<< <i>inout</i> >> (p. 1013) The request data
<i>request_info</i>	<< <i>inout</i> >> (p. 1013) Optional write parameters. When this call ends succesfully,
Generated by Doxygen	DDS_WriteParams_t::identity (p. 2102) contains a valid identity that can be used for correlation with future replies.

Returns

One of the **Standard Return Codes** (p. 1221) ; **DDS_RETCODE_TIMEOUT** (p. 1222) may be reported in the same conditions as in **FooDataWriter_write** (p. 496).

MT Safety:

SAFE

See also

DDS_WriteParams_t::identity (p. 2102)
RTI_Connext_Requester_wait_for_replies_for_related_request (p. 751)
FooBarRequester_take_replies_for_related_request (p. 757)
Correlating requests and replies (p. 1006)

5.42.4.8 FooBarRequester_receive_reply()

```
DDS_ReturnCode_t FooBarRequester_receive_reply (
    FooBarRequester * self,
    Bar * reply,
    struct DDS_SampleInfo * sample_info,
    const struct DDS_Duration_t * max_wait )
```

Waits for a reply and copies its contents into a Sample.

This operation is equivalent to using **RTI_Connext_Requester_wait_for_replies** (p. 750) and **FooBarRequester_↵take_reply** (p. 755).

MT Safety:

Same restrictions as **RTI_Connext_Requester_wait_for_replies** (p. 750)

See also

RTI_Connext_Requester_wait_for_replies (p. 750)
FooBarRequester_take_reply (p. 755)
Requester example (p. 1005)

5.42.4.9 FooBarRequester_receive_replies()

```
DDS_ReturnCode_t FooBarRequester_receive_replies (
    FooBarRequester * self,
    struct BarSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long min_count,
    DDS_Long max_count,
    const struct DDS_Duration_t * max_wait )
```

Waits for multiple replies and provides a loaned sequence to access them.

This operation is equivalent to using **RTI_Connext_Requester_wait_for_replies** (p. 750) and **FooBarRequester_take_replies** (p. 756).

MT Safety:

See **RTI_Connext_Requester_wait_for_replies** (p. 750)

Returns

One of the **Standard Return Codes** (p. 1221)

See also

RTI_Connext_Requester_wait_for_replies (p. 750)

FooBarRequester_take_replies (p. 756)

5.42.4.10 FooBarRequester_take_reply()

```
DDS_ReturnCode_t FooBarRequester_take_reply (
    FooBarRequester * self,
    Bar * reply,
    struct DDS_SampleInfo * sample_info )
```

Copies the contents of a reply into a Sample.

Takes a reply sample from the Requester and makes a copy.

This operation may be used after a call to **RTI_Connext_Requester_wait_for_replies** (p. 750).

To avoid copies, you can use **FooBarRequester_take_replies** (p. 756).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>reply</i>	<< <i>inout</i> >> (p. 1013) user data type-specific Foo (p. 2109) object where the next received reply sample will be returned. The received_data must have been fully allocated. Otherwise, this operation may fail.
<i>sample_info</i>	<< <i>inout</i> >> (p. 1013) a DDS_SampleInfo (p. 1985) object where the info associated to the received reply will be returned. Must be a valid non-NULL DDS_SampleInfo (p. 1985). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.

Returns

One of the **Standard Return Codes** (p. 1221) , **DDS_RETCODE_NO_DATA** (p. 1222)

MT Safety:

SAFE

See also

RTI_Connext_Requester_wait_for_replies (p. 750)

5.42.4.11 FooBarRequester_take_replies()

```
DDS_ReturnCode_t FooBarRequester_take_replies (
    FooBarRequester * self,
    struct BarSeq * reply_seq,
    struct DDS_SampleInfoSeq * sample_info_seq,
    DDS_Long max_count )
```

Provides a loaned sequence to access the existing replies.

Takes all the existing replies up to `max_count` and provides a loaned sequence to access them.

This operation does not make a copy of the data. It is similar to **FooDataReader_take** (p. 626).

The loan is returned with **FooBarRequester_return_loan** (p. 760)

This operation may be used after a call to **RTI_Connext_Requester_wait_for_replies** (p. 750)

See an **example** here: **Taking loaned samples** (p. 1005)

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>reply_seq</i>	The data sequence
<i>sample_info_seq</i>	The associated sample info sequence
<i>max_count</i>	The maximum number of samples that are taken at a time. The special value DDS_LENGTH_UNLIMITED (p. 1326) may be used. This value will read up to the limit specified by DDS_DataReaderResourceLimitsQosPolicy::max_samples_per_read (p. 1625)

Returns

One of the **Standard Return Codes** (p. 1221) , **DDS_RETCODE_NO_DATA** (p. 1222)

MT Safety:

SAFE

See also

RTI_Connext_Requester_wait_for_replies (p. 750)

FooDataReader_take (p. 626) (for a more detailed description on how QoS and other parameters affect the underlying DataReader)

5.42.4.12 FooBarRequester_take_reply_for_related_request()

```
DDS_ReturnCode_t FooBarRequester_take_reply_for_related_request (
    FooBarRequester * self,
    Bar * reply,
    struct DDS_SampleInfo * sample_info,
    const struct DDS_SampleIdentity_t * related_request_id )
```

Copies the contents of a reply for a specific request.

This operation is analogous to **FooBarRequester_take_reply** (p. 755) except the reply corresponds to a specific request.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>reply</i>	The sample where a reply is copied into
<i>sample_info</i>	<< inout >> (p. 1013) a DDS_SampleInfo (p. 1985) object where the info associated to the received reply will be returned. Must be a valid non-NULL DDS_SampleInfo (p. 1985). The function will fail with DDS_RETCODE_BAD_PARAMETER (p. 1222) if it is NULL.
<i>related_request_id</i>	The identity of a request previously sent by this Requester

MT Safety:

SAFE

See also

FooBarRequester_take_reply (p. 755)

FooBarRequester_send_request_w_params (p. 753)

Correlating requests and replies (p. 1006)

5.42.4.13 FooBarRequester_take_replies_for_related_request()

```
DDS_ReturnCode_t FooBarRequester_take_replies_for_related_request (
    FooBarRequester * self,
    struct BarSeq * reply_seq,
    struct DDS_SampleInfoSeq * sample_info_seq,
    DDS_Long max_count,
    const struct DDS_SampleIdentity_t * related_request_id )
```

Provides a loaned sequence to access the existing replies for a specific request.

This operation is analogous to **FooBarRequester_take_replies** (p. 756) except the replies this operation provides correspond to a specific request.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>reply_seq</i>	The data sequence
<i>sample_info_seq</i>	The associated sample info sequence
<i>max_count</i>	The maximum number of samples that are taken at a time. The special value DDS_LENGTH_UNLIMITED (p. 1326) may be used.
<i>related_request_id</i>	The identity of a request previously sent by this Requester

MT Safety:

SAFE

See also

FooBarRequester_take_replies (p. 756)

Correlating requests and replies (p. 1006)

5.42.4.14 FooBarRequester_read_reply()

```
DDS_ReturnCode_t FooBarRequester_read_reply (
    FooBarRequester * self,
    Bar * reply,
    struct DDS_SampleInfo * sample_info )
```

Copies the contents of a reply into a Sample.

This operation is equivalent to **FooBarRequester_take_reply** (p. 755) except the reply remains in the Requester and can be read or taken again.

5.42.4.15 FooBarRequester_read_replies()

```
DDS_ReturnCode_t FooBarRequester_read_replies (
    FooBarRequester * self,
    struct BarSeq * reply_seq,
    struct DDS_SampleInfoSeq * sample_info_seq,
    DDS_Long max_count )
```

Provides a loaned sequence to access the existing replies.

This operation is equivalent to **FooBarRequester_take_replies** (p. 756) except the replies remain in the Requester and can be read or taken again.

5.42.4.16 FooBarRequester_read_reply_for_related_request()

```
DDS_ReturnCode_t FooBarRequester_read_reply_for_related_request (
    FooBarRequester * self,
    Bar * reply,
    struct DDS_SampleInfo * sample_info,
    const struct DDS_SampleIdentity_t * related_request_id )
```

Copies the contents of a reply for a specific request.

This operation is equivalent to **FooBarRequester_take_reply_for_related_request** (p. 757) except the reply remains in the Requester and can be read or taken again.

5.42.4.17 FooBarRequester_read_replies_for_related_request()

```
DDS_ReturnCode_t FooBarRequester_read_replies_for_related_request (
    FooBarRequester * self,
    struct BarSeq * reply_seq,
    struct DDS_SampleInfoSeq * sample_info_seq,
    DDS_Long max_count,
    const struct DDS_SampleIdentity_t * related_request_id )
```

Provides a loaned sequence to access the existing replies for a specific request.

This operation is equivalent to **FooBarRequester_take_replies_for_related_request** (p. 757) except the replies remain in the Requester and can be read or taken again.

5.42.4.18 FooBarRequester_get_request_datawriter()

```
FooDataWriter * FooBarRequester_get_request_datawriter (
    FooBarRequester * self )
```

Retrieves the underlying **DDS_DataWriter** (p. 485).

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

DDS_DataWriter (p. 485)
FooDataWriter (p. 2113)
DDS_DataWriter_get_matched_subscriptions (p. 539)
DDS_DataWriter_get_matched_subscription_data (p. 540)
DDS_DataWriter_set_listener (p. 553)
DDS_DataWriter_get_datawriter_protocol_status (p. 548)

5.42.4.19 FooBarRequester_get_reply_datareader()

```
BarDataReader * FooBarRequester_get_reply_datareader (
    FooBarRequester * self )
```

Retrieves the underlying **DDS_DataReader** (p. 615).

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

DDS_DataReader (p. 615)
FooDataReader (p. 2112)
DDS_DataReader_get_matched_publications (p. 678)
DDS_DataReader_get_matched_publication_data (p. 679)
DDS_DataReader_set_listener (p. 690)
DDS_DataReader_get_datareader_protocol_status (p. 686)

5.42.4.20 FooBarRequester_return_loan()

```
DDS_ReturnCode_t FooBarRequester_return_loan (
    FooBarRequester * self,
    struct BarSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq )
```

Returns samples previously received from the middleware.

This operation needs to be called at some point after:

- **FooBarRequester_take_replies** (p. 756)
- **FooBarRequester_take_replies_for_related_request** (p. 757)
- **FooBarRequester_receive_replies** (p. 754)

See also

FooDataReader_return_loan (p. 646), for more information on how the middleware loans data to the application.

5.43 Replier

FooBarReplier (p. 2110), **FooBarSimpleReplier** (p. 2112) and associated elements

Data Structures

- struct **RTI_Connext_ReplierListener**
*Called when a **FooBarReplier** (p. 2110) has new available requests.*
- struct **RTI_Connext_SimpleReplierListener**
*The listener called by a **SimpleReplier**.*
- struct **RTI_Connext_Replier**
*The type-independent version of a **Replier**.*
- struct **RTI_Connext_ReplierParams**
*Contains the parameters for creating a **FooBarReplier** (p. 2110).*
- struct **FooBarReplier**
Allows receiving requests and sending replies.
- struct **RTI_Connext_SimpleReplierParams**
*Contains the parameters for creating a **FooBarSimpleReplier** (p. 2112).*
- struct **FooBarSimpleReplier**
A callback-based repplier.

Macros

- **#define RTI_Connext_ReplierListener_INITIALIZER {NULL, NULL}**
*Initializes an **RTI_Connext_ReplierListener** (p. 2168).*
- **#define RTI_Connext_ReplierParams_INITIALIZER**
*Initializes a **RTI_Connext_ReplierParams** (p. 2169) instance.*
- **#define RTI_CONNEXT_REPLIER_DECL(TReq, TRep, TReplier)**
Instantiates the declaration of a typed replier and its operations.
- **#define RTI_Connext_RequesterParams_INITIALIZER**
*Initializes a **RTI_Connext_RequesterParams** (p. 2173) instance.*
- **#define RTI_Connext_SimpleReplierParams_INITIALIZER**
*Initializes a **RTI_Connext_SimpleReplierParams** (p. 2177) instance.*
- **#define RTI_CONNEXT_SIMPLEREPLIER_DECL(TReq, TRep, TSimpleReplier)**
Instantiates the declaration of a typed SimpleReplier and its operations.

Typedefs

- **typedef void(* RTI_Connext_ReplierListener_OnRequestAvailableCallback) (struct RTI_Connext_ReplierListener *self, RTI_Connext_Replier *replier)**
*The type of **RTI_Connext_ReplierListener::on_request_available** (p. 774).*
- **typedef void>(* RTI_Connext_SimpleReplierListener_OnRequestAvailableCallback) (RTI_Connext_SimpleReplierListener *self, const void *request, const struct DDS_SampleInfo *info)**
*The type of **RTI_Connext_SimpleReplierListener::on_request_available** (p. 775).*
- **typedef void(* RTI_Connext_SimpleReplierListener_OnReturnLoanCallback) (RTI_Connext_SimpleReplierListener *self, void *reply)**
*The type of **RTI_Connext_SimpleReplierListener::return_loan** (p. 775).*
- **typedef struct RTI_Connext_ReplierParams RTI_Connext_ReplierParams**
*Contains the parameters for creating a **FooBarReplier** (p. 2110).*
- **typedef struct FooBarReplier FooBarReplier**
Allows receiving requests and sending replies.
- **typedef struct RTI_Connext_SimpleReplierParams RTI_Connext_SimpleReplierParams**
*Contains the parameters for creating a **FooBarSimpleReplier** (p. 2112).*
- **typedef struct FooBarSimpleReplier FooBarSimpleReplier**
A callback-based replier.

Functions

- **DDS_ReturnCode_t RTI_Connext_Replier_delete (RTI_Connext_Replier *self)**
Releases the internal entities created by this Replier.
- **DDS_ReturnCode_t RTI_Connext_Replier_wait_for_requests (RTI_Connext_Replier *self, int min_count, const struct DDS_Duration_t *max_wait)**
Waits for requests.
- **FooBarReplier * FooBarReplier_create (DDS_DomainParticipant *participant, char *service_name)**
Creates a Replier with the minimum set of parameters.
- **FooBarReplier * FooBarReplier_create_w_params (const RTI_Connext_ReplierParams *params)**
Creates a Replier with parameters.

- **DDS_ReturnCode_t FooBarReplier_receive_request** (**FooBarReplier** *self, **Foo** *request, struct **DDS_SampleInfo** *sample_info, const struct **DDS_Duration_t** *max_wait)
Waits for a request and copies its contents into a Sample.
- **DDS_ReturnCode_t FooBarReplier_receive_requests** (**FooBarReplier** *self, struct **FooSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** min_reply_count, **DDS_Long** max_reply_count, const struct **DDS_Duration_t** *max_wait)
Waits for multiple requests and provides a loaned sequence to access them.
- **DDS_ReturnCode_t FooBarReplier_take_request** (**FooBarReplier** *self, **Foo** *request, struct **DDS_SampleInfo** *sample_info)
Copies the contents of a request into a Sample.
- **DDS_ReturnCode_t FooBarReplier_take_requests** (**FooBarReplier** *self, struct **FooSeq** *request_seq, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_count)
Provides a loaned sequence to access the existing requests.
- **DDS_ReturnCode_t FooBarReplier_read_request** (**FooBarReplier** *self, **Foo** *request, struct **DDS_SampleInfo** *sample_info)
Copies the contents of a request into a Sample.
- **DDS_ReturnCode_t FooBarReplier_read_requests** (**FooBarReplier** *self, struct **FooSeq** *request_seq, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_count)
Provides a loaned sequence to access the existing requests.
- **DDS_ReturnCode_t FooBarReplier_send_reply** (**FooBarReplier** *self, **Bar** *reply, const struct **DDS_SampleIdentity_t** *related_request_id)
Sends a reply for a previous request.
- **FooDataReader * FooBarReplier_get_request_datareader** (**FooBarReplier** *self)
*Retrieves the underlying **DDS_DataReader** (p. 615).*
- **BarDataWriter * FooBarReplier_get_reply_datawriter** (**FooBarReplier** *self)
*Retrieves the underlying **DDS_DataWriter** (p. 485).*
- **DDS_ReturnCode_t FooBarReplier_return_loan** (**FooBarReplier** *self, struct **FooSeq** *replies, struct **DDS_SampleInfoSeq** *info_seq)
Returns samples previously received from the middleware.
- **FooBarSimpleReplier * FooBarSimpleReplier_create** (**DDS_DomainParticipant** *participant, char *service_name, **RTI_Connext_SimpleReplierListener** *listener)
Creates a new SimpleReplier.
- **FooBarSimpleReplier * FooBarSimpleReplier_create_w_params** (**RTI_Connext_SimpleReplierParams** *params)
Creates a new SimpleReplier.
- **DDS_ReturnCode_t FooBarSimpleReplier_delete** (**FooBarSimpleReplier** *self)
Releases the resources created by this SimpleReplier.

Variables

- **RTI_Connext_ReplierListener_OnRequestAvailableCallback request_available** **RTI_Connext_ReplierListener::on_request_available**
User callback.
- **RTI_Connext_SimpleReplierListener_OnRequestAvailableCallback Listener::on_request_available** **RTI_Connext_SimpleReplierListener::on_request_available**
User callback that receives a request and provides a reply.
- **RTI_Connext_SimpleReplierListener_OnReturnLoanCallback ::return_loan** **RTI_Connext_SimpleReplierListener::return_loan**
Returns a previously generated reply to the user.

5.43.1 Detailed Description

FooBarReplier (p. 2110), **FooBarSimpleReplier** (p. 2112) and associated elements

5.43.2 Macro Definition Documentation

5.43.2.1 RTI_Connext_ReplierListener_INITIALIZER

```
#define RTI_Connext_ReplierListener_INITIALIZER {NULL, NULL}
```

Intializes an **RTI_Connext_ReplierListener** (p. 2168).

5.43.2.2 RTI_Connext_ReplierParams_INITIALIZER

```
#define RTI_Connext_ReplierParams_INITIALIZER
```

Initializes a **RTI_Connext_ReplierParams** (p. 2169) instance.

5.43.2.3 RTI_CONNEXT_REPLIER_DECL

```
#define RTI_CONNEXT_REPLIER_DECL(  
    TReq,  
    TRep,  
    TReplier )
```

Instantiates the declaration of a typed replier and its operations.

Parameters

<i>TReq</i>	The request type name
<i>TRep</i>	The reply type name
<i>TReplier</i>	The type name for the typed replier and the prefix for all the operations.

See also

Creating a Replier (p. 1007)

5.43.2.4 RTI_Connext_RequesterParams_INITIALIZER

```
#define RTI_Connext_RequesterParams_INITIALIZER
```

Initializes a **RTI_Connext_RequesterParams** (p. 2173) instance.

5.43.2.5 RTI_Connext_SimpleReplierParams_INITIALIZER

```
#define RTI_Connext_SimpleReplierParams_INITIALIZER
```

Initializes a **RTI_Connext_SimpleReplierParams** (p. 2177) instance.

5.43.2.6 RTI_CONNEXT_SIMPLEREPLIER_DECL

```
#define RTI_CONNEXT_SIMPLEREPLIER_DECL(  
    TReq,  
    TRep,  
    TSimpleReplier )
```

Instantiates the declaration of a typed SimpleReplier and its operations.

Parameters

<i>TReq</i>	The request type name
<i>TRep</i>	The reply type name
<i>TSimpleReplier</i>	The type name for the typed SimpleReplier and the prefix for all the operations.

See also

SimpleReplier example (p. 1009)

5.43.3 Typedef Documentation

5.43.3.1 RTI_Connext_ReplierListener_OnRequestAvailableCallback

```
typedef void(* RTI_Connext_ReplierListener_OnRequestAvailableCallback) (struct RTI_Connext_↔  
ReplierListener *self, RTI_Connext_Replier *replier)
```

The type of **RTI_Connext_ReplierListener::on_request_available** (p. 774).

5.43.3.2 RTI_Connext_SimpleReplierListener_OnRequestAvailableCallback

```
typedef void (* RTI_Connext_SimpleReplierListener_OnRequestAvailableCallback) ( RTI_Connext_↵
SimpleReplierListener *self, const void *request, const struct DDS_SampleInfo *info)
```

The type of `RTI_Connext_SimpleReplierListener::on_request_available` (p. 775).

5.43.3.3 RTI_Connext_SimpleReplierListener_OnReturnLoanCallback

```
typedef void(* RTI_Connext_SimpleReplierListener_OnReturnLoanCallback) ( RTI_Connext_Simple↵
ReplierListener *self, void *reply)
```

The type of `RTI_Connext_SimpleReplierListener::return_loan` (p. 775).

5.43.3.4 RTI_Connext_ReplierParams

```
typedef struct RTI_Connext_ReplierParams RTI_Connext_ReplierParams
```

Contains the parameters for creating a **FooBarReplier** (p. 2110).

See also

RTI_Connext_RequesterParams (p. 2173)

5.43.3.5 FooBarReplier

```
typedef struct FooBarReplier FooBarReplier
```

Allows receiving requests and sending replies.

A Replier is an entity with two associated **topics** (p. 167): 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. 305), and **DDS_DynamicData** (p. 1768). **Note:** At this moment, in the C version of this API, only `rtiddsgen`-generated types are supported.

To create a Replier for two types, a request type `TReq=Foo` (p. 2109) and a reply type `TRep=Bar`, your application needs to instantiate the data structure **FooBarReplier** (p. 2110) and the specific operations that can publish and subscribe to those types. In this documentation we refer to the type-dependent operations as `FooBarReplier_` (for example, `FooBarReplier_take_requests` (p. 771)). Some operations are type-independent and their names always begin with `RTI_Connext_Replier_` (for example, `RTI_Connext_Replier_wait_for_requests` (p. 768)). See **Creating a Replier** (p. 1007).

A Replier 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 original Requester can identify it)

For multi-reply scenarios in which a **FooBarReplier** (p. 2110) generates more than one reply for a request, the **FooBarReplier** (p. 2110) should mark all the intermediate replies (all but the last) using the **DDS_INTERMEDIATE_REPLY_SEQUENCE_SAMPLE** (p. 1388) flag in **DDS_WriteParams_t::flag** (p. 2104).

Much like a Requester, a Replier has an associated **DDS_DomainParticipant** (p. 77), which can be shared with other Repliers or RTI Connex routines. All the other entities required for the request-reply interaction, including a **DDS_DataWriter** (p. 485) for writing replies and a **DDS_DataReader** (p. 615) for reading requests, are automatically created when the Replier is constructed.

Quality of Service for the underlying DataWriter and DataReader can be configured (see **RTI_Connext_Requester_Params::qos_profile_name** (p. 2175)). By default, they are created with **DDS_RELIABLE_RELIABILITY_QOS** (p. 1323). The exact default configuration is described here: **Configuring Request-Reply QoS profiles** (p. 1010)

There are several ways to use a Replier:

- A thread **receives** (p. 770) requests and then dispatches them. If the computation of a reply is a simple operation, consider using a **FooBarSimpleReplier** (p. 2112) instead of a Replier.
- Polling without waiting, using **FooBarReplier_take_requests** (p. 771) directly.
- Using a **RTI_Connext_ReplierListener** (p. 2168) to get notified and **get** (p. 771) the requests within the callback.

See also

FooBarRequester (p. 2111)

Request-Reply Examples (p. 1002)

Replier example (p. 1008)

5.43.3.6 RTI_Connext_SimpleReplierParams

```
typedef struct RTI_Connext_SimpleReplierParams RTI_Connext_SimpleReplierParams
```

Contains the parameters for creating a **FooBarSimpleReplier** (p. 2112).

The parameters for a SimpleReplier are identical to those of the Replier, except for the SimpleReplierListener, which is required and has a different user callback.

See also

RTI_Connext_ReplierParams (p. 2169)

5.43.3.7 FooBarSimpleReplier

```
typedef struct FooBarSimpleReplier FooBarSimpleReplier
```

A callback-based replier.

A SimpleReplier is based on a **RTI_Connext_SimpleReplierListener** (p.2176) that users provide . Requests are passed to the callback, which returns a reply. The reply is directed only to the Requester 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 **FooBarReplier** (p.2110) instead.

See also

FooBarReplier (p.2110)

RTI_Connext_SimpleReplierListener (p.2176)

SimpleReplier example (p.1009)

5.43.4 Function Documentation

5.43.4.1 RTI_Connext_Replier_delete()

```
DDS_ReturnCode_t RTI_Connext_Replier_delete (
    RTI_Connext_Replier * self )
```

Releases the internal entities created by this Replier.

Among other internal resources, it deletes the Replier's underlying DataReader and DataWriter.

See also

DDS_Subscriber_delete_datareader (p.585)

DDS_Publisher_delete_datawriter (p.456)

5.43.4.2 RTI_Connext_Replier_wait_for_requests()

```
DDS_ReturnCode_t RTI_Connext_Replier_wait_for_requests (
    RTI_Connext_Replier * self,
    int min_count,
    const struct 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 **RTI_Connext_Requester_wait_for_replies** (p.750).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>min_count</i>	Minimum number of requests that need to be available for this operation to unblock.
<i>max_wait</i>	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

FooBarReplier_take_requests (p. 771)

RTI_Connext_Requester_wait_for_replies (p. 750)

5.43.4.3 FooBarReplier_create()

```

FooBarReplier * FooBarReplier_create (
    DDS_DomainParticipant * participant,
    char * service_name )

```

Creates a Replier with the minimum set of parameters.

Parameters

<i>participant</i>	The DomainParticipant that this Replier uses to join a domain.
<i>service_name</i>	The service name. See RTI_Connext_ReplierParams::service_name (p. 2170)

Returns

One of the **Standard Return Codes** (p. 1221)

5.43.4.4 FooBarReplier_create_w_params()

```

FooBarReplier * FooBarReplier_create_w_params (
    const RTI_Connext_ReplierParams * params )

```

Creates a Replier with parameters.

Parameters

<i>params</i>	All the parameters that configure this Replier

Returns

One of the **Standard Return Codes** (p. 1221)

See also

RTI_Connext_ReplierParams (p. 2169)

Creating a Replier (p. 1007)

5.43.4.5 FooBarReplier_receive_request()

```
DDS_ReturnCode_t FooBarReplier_receive_request (
    FooBarReplier * self,
    Foo * request,
    struct DDS_SampleInfo * sample_info,
    const struct DDS_Duration_t * max_wait )
```

Waits for a request and copies its contents into a Sample.

Equivalent to using **RTI_Connext_Replier_wait_for_requests** (p. 768) and **FooBarReplier_take_request** (p. 770)

See also

RTI_Connext_Replier_wait_for_requests (p. 768)

FooBarReplier_take_request (p. 770)

Replier example (p. 1008)

5.43.4.6 FooBarReplier_receive_requests()

```
DDS_ReturnCode_t FooBarReplier_receive_requests (
    FooBarReplier * self,
    struct FooSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long min_reply_count,
    DDS_Long max_reply_count,
    const struct DDS_Duration_t * max_wait )
```

Waits for multiple requests and provides a loaned sequence to access them.

Equivalent to using **RTI_Connext_Replier_wait_for_requests** (p. 768) and **FooBarReplier_take_requests** (p. 771)

See also

RTI_Connext_Replier_wait_for_requests (p. 768)

FooBarReplier_take_requests (p. 771)

5.43.4.7 FooBarReplier_take_request()

```
DDS_ReturnCode_t FooBarReplier_take_request (
    FooBarReplier * self,
    Foo * request,
    struct DDS_SampleInfo * sample_info )
```

Copies the contents of a request into a Sample.

See also

FooBarRequester_take_reply (p. 755)

5.43.4.8 FooBarReplier_take_requests()

```
DDS_ReturnCode_t FooBarReplier_take_requests (
    FooBarReplier * self,
    struct FooSeq * request_seq,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_count )
```

Provides a loaned sequence to access the existing requests.

See also

FooBarRequester_take_replies (p. 756)

5.43.4.9 FooBarReplier_read_request()

```
DDS_ReturnCode_t FooBarReplier_read_request (
    FooBarReplier * self,
    Foo * request,
    struct DDS_SampleInfo * sample_info )
```

Copies the contents of a request into a Sample.

This operation is equivalent to **FooBarReplier_take_request** (p. 770) except the request remains in the Replier and can be read or taken again.

5.43.4.10 FooBarReplier_read_requests()

```
DDS_ReturnCode_t FooBarReplier_read_requests (
    FooBarReplier * self,
    struct FooSeq * request_seq,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_count )
```

Provides a loaned sequence to access the existing requests.

This operation is equivalent to **FooBarReplier_take_requests** (p. 771) except the requests remain in the Replier and can be read or taken again.

5.43.4.11 FooBarReplier_send_reply()

```
DDS_ReturnCode_t FooBarReplier_send_reply (
    FooBarReplier * self,
    Bar * reply,
    const struct DDS_SampleIdentity_t * related_request_id )
```

Sends a reply for a previous request.

The related request identity can be retrieved from the sample info (**DDS_SampleInfo_get_sample_identity** (p. 702))

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>reply</i>	The reply to be sent.
<i>related_request_id</i>	The identity of a previously received request

Returns

One of the **Standard Return Codes** (p. 1221)

See also

FooBarReplier_receive_request (p. 770)
FooBarReplier_receive_requests (p. 770)
FooBarReplier_take_request (p. 770)
FooBarReplier_take_requests (p. 771)
Replier example (p. 1008)

5.43.4.12 FooBarReplier_get_request_datareader()

```
FooDataReader * FooBarReplier_get_request_datareader (
    FooBarReplier * self )
```

Retrieves the underlying **DDS_DataReader** (p. 615).

See also

FooBarRequester_get_reply_datareader (p. 760)

5.43.4.13 FooBarReplier_get_reply_datawriter()

```
BarDataWriter * FooBarReplier_get_reply_datawriter (
    FooBarReplier * self )
```

Retrieves the underlying **DDS_DataWriter** (p. 485).

See also

FooBarRequester_get_request_datawriter (p. 759)

5.43.4.14 FooBarReplier_return_loan()

```
DDS_ReturnCode_t FooBarReplier_return_loan (
    FooBarReplier * self,
    struct FooSeq * replies,
    struct DDS_SampleInfoSeq * info_seq )
```

Returns samples previously received from the middleware.

This operation needs to be called at some point after:

- **FooBarReplier_take_requests** (p. 771)
- **FooBarReplier_receive_requests** (p. 770)

See also

FooDataReader_return_loan (p. 646), for more information on how the middleware loans data to the application.

5.43.4.15 FooBarSimpleReplier_create()

```
FooBarSimpleReplier * FooBarSimpleReplier_create (
    DDS_DomainParticipant * participant,
    char * service_name,
    RTI_Connext_SimpleReplierListener * listener )
```

Creates a new SimpleReplier.

See also

[RTI_Connext_SimpleReplierParams](#) (p. 2177)

[RTI_Connext_SimpleReplierListener](#) (p. 2176)

5.43.4.16 FooBarSimpleReplier_create_w_params()

```
FooBarSimpleReplier * FooBarSimpleReplier_create_w_params (
    RTI_Connext_SimpleReplierParams * params )
```

Creates a new SimpleReplier.

See also

[RTI_Connext_SimpleReplierParams](#) (p. 2177)

[RTI_Connext_SimpleReplierListener](#) (p. 2176)

5.43.4.17 FooBarSimpleReplier_delete()

```
DDS_ReturnCode_t FooBarSimpleReplier_delete (
    FooBarSimpleReplier * self )
```

Releases the resources created by this SimpleReplier.

See also

[RTI_Connext_Replier_delete](#) (p. 768)

5.43.5 Variable Documentation

5.43.5.1 on_request_available [1/2]

RTI_Connext_ReplierListener_OnRequestAvailableCallback `RTI_Connext_ReplierListener::on_request_available`

User callback.

See also

DDS_DataReaderListener::on_data_available (p. 1595)

5.43.5.2 on_request_available [2/2]

RTI_Connext_SimpleReplierListener_OnRequestAvailableCallback `RTI_Connext_SimpleReplierListener::on_request_available`

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, **RTI_Connext_SimpleReplierListener::return_loan** (p. 775) is called.

Parameters

<i>request</i>	The received request

Returns

A reply for that request

5.43.5.3 return_loan

RTI_Connext_SimpleReplierListener_OnReturnLoanCallback `RTI_Connext_SimpleReplierListener::return_loan`

Returns a previously generated reply to the user.

This operation is always called right after sending the reply created by **RTI_Connext_SimpleReplierListener::on_request_available** (p. 775). 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

<i>reply</i>	The reply previously provided in RTI_Connext_SimpleReplierListener::on_request_available (p. 775)

5.44 Utilities

Utilities for the RTI Connex Messaging module.

Typedefs

- typedef struct **DDS_ProductVersion_t RTI_Connext_Messaging_LibraryVersion**
This struct represents the library version.

Functions

- const **RTI_Connext_Messaging_LibraryVersion * RTI_Connext_Messaging_get_api_version** (void)
Get the API version number.
- const char * **RTI_Connext_Messaging_Library_get_api_version_string** (void)
Get the API version strings.
- const char * **RTI_Connext_Messaging_get_api_build_number_string** (void)
Get the build number string.

5.44.1 Detailed Description

Utilities for the RTI Connex Messaging module.

Connex Messaging Utilities

5.44.2 Typedef Documentation

5.44.2.1 RTI_Connext_Messaging_LibraryVersion

```
typedef struct DDS_ProductVersion_t RTI_Connext_Messaging_LibraryVersion
```

This struct represents the library version.

5.44.3 Function Documentation

5.44.3.1 RTI_Connext_Messaging_get_api_version()

```
const RTI_Connext_Messaging_LibraryVersion * RTI_Connext_Messaging_get_api_version (
    void )
```

Get the API version number.

5.44.3.2 RTI_Connext_Messaging_Library_get_api_version_string()

```
const char * RTI_Connext_Messaging_Library_get_api_version_string (
    void )
```

Get the API version strings.

5.44.3.3 RTI_Connext_Messaging_get_api_build_number_string()

```
const char * RTI_Connext_Messaging_get_api_build_number_string (
    void )
```

Get the build number string.

5.45 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. 778)
- **Durable Reader State** (p. 778)
- **Data Durability** (p. 778)

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.

5.45.1 Durable Writer History

This feature allows a **DDS_DataWriter** (p. 485) to locally persist its local history cache so that it can survive shutdowns, crashes and restarts. When an application restarts, each **DDS_DataWriter** (p. 485) 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 **DDS_DataWriter** (p. 485) had been temporarily disconnected from the network and then reappeared.

See also

Configuring Durable Writer History (p. 779)

5.45.2 Durable Reader State

This feature allows a **DDS_DataReader** (p. 615) to locally persist its state and remember the sequence numbers it has already received. When an application restarts, each **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615) before the restart will not be provided to the application again.

See also

Configuring Durable Reader State (p. 781)

5.45.3 Data Durability

This feature is a full implementation of the OMG DDS Persistence Profile. The **DURABILITY** (p. 1285) QoS lets an application configure a **DDS_DataWriter** (p. 485) such that the information written by the **DDS_DataWriter** (p. 485) survives beyond the lifetime of the **DDS_DataWriter** (p. 485). In this manner, a late-joining **DDS_DataReader** (p. 615) can subscribe and receive the information even after the **DDS_DataWriter** (p. 485) application is no longer executing. To use this feature, you need RTI Persistence Service – an optional product that can be purchased separately.

5.45.4 Durability and Persistence Based on Virtual GUID

Every modification to the global dataspace made by a **DDS_DataWriter** (p. 485) is identified by a pair (virtual GUID, sequence number).

- The virtual GUID (Global Unique Identifier) is a 16-byte character identifier associated with a **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615); 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 **DDS_DataWriter** (p. 485).

Several **DDS_DataWriter** (p. 485) entities can be configured with the same virtual GUID. If each of these **DDS_DataWriter** (p. 485) entities publishes a sample with sequence number '0', the sample will only be received once by the **DDS_DataReader** (p. 615) entities subscribing to the content published by the **DDS_DataWriter** (p. 485) 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 **DDS_DataWriter** (p. 485) will be persisted in a database table with a name generated from the virtual GUID of the **DDS_DataWriter** (p. 485). If the **DDS_DataWriter** (p. 485) is restarted, it must have associated the same virtual GUID to restore its previous history.

Likewise, the state of a **DDS_DataReader** (p. 615) will be persisted in a database table whose name is generated from the **DDS_DataReader** (p. 615) virtual GUID

A **DDS_DataWriter** (p. 485)'s virtual GUID can be configured using **DDS_DataWriterProtocolQosPolicy::virtual_guid** (p. 1650). Similarly, a **DDS_DataReader** (p. 615)'s virtual GUID can be configured using **DDS_DataReaderProtocolQosPolicy::virtual_guid** (p. 1597).

The **DDS_PublicationBuiltinTopicData** (p. 1905) and **DDS_SubscriptionBuiltinTopicData** (p. 2012) 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. 1650) **DDS_DataReaderProtocolQosPolicy::virtual_guid** (p. 1597).

5.45.5 Configuring Durable Writer History

To configure a **DDS_DataWriter** (p. 485) to have durable writer history, use the **PROPERTY** (p. 1306) QoS policy associated with the **DDS_DataWriter** (p. 485) or the **DDS_DomainParticipant** (p. 77).

Properties defined for the **DDS_DomainParticipant** (p. 77) will be applied to all the **DDS_DataWriter** (p. 485) objects belonging to the **DDS_DomainParticipant** (p. 77), unless the property is overwritten by the **DDS_DataWriter** (p. 485).

See also

DDS_PropertyQosPolicy (p. 1901)

The following table lists the supported durable writer history properties.

Table 5.743 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 DDS_DataWriter (p. 485) will create its own database connection. Default: 0
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_ResourceLimitsQosPolicy (p. 1949) are used to configure the maximum number of samples and instances that can be stored in the relational database. Default: DDS_ResourceLimitsQosPolicy::max_instances (p. 1951)
dds.data_writer.history.odbc_plugin.builtin.instance_cache_init_size	See description above. Default: DDS_ResourceLimitsQosPolicy::initial_instances (p. 1952)
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 DDS_DataWriter (p. 485) is restarted. If the value is 0, the content of the database associated with the DDS_DataWriter (p. 485) being restarted will be deleted. If the value is 1, the DDS_DataWriter (p. 485) will restore its previous state from the database content. Default: 1

Property	Description
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 <code>in_memory_state</code> is equal to 1, <code>instance_cache_max_size</code> is always equal to DDS_ResourceLimitsQosPolicy::max_instances (p. 1951) (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 <code>in_memory_state</code> 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 (TimesTen). Default: 1

5.45.6 Configuring Durable Reader State

To configure a **DDS_DataReader** (p. 615) with durable reader state, use the **PROPERTY** (p. 1306) QoS policy associated with the **DDS_DataReader** (p. 615) or **DDS_DomainParticipant** (p. 77).

A property defined in the **DDS_DomainParticipant** (p. 77) will be applicable to all the **DDS_DataReader** (p. 615) belonging to the **DDS_DomainParticipant** (p. 77) unless it is overwritten by the **DDS_DataReader** (p. 615).

See also

DDS_PropertyQosPolicy (p. 1901)

The following table lists the supported durable reader state properties.

Table 5.744 Durable Reader State Properties

Property	Description
dds.data_reader.state.odbc.dsn	The ODBC DSN (Data Source Name) associated with the database where the DDS_DataReader (p. 615) 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.

Property	Description
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 DDS_Data ↵ Reader (p.615) state must be restored or not once the DDS_DataReader (p.615) is restarted. If this property is 0, the previous state will be deleted from the database. If it is 1, the DDS_DataReader (p.615) 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
dds.data_reader.state.persistence_service.request_↵ depth	This property indicates the number of most recent historical samples that the persisted DDS_DataReader (p.615) wants to receive when it starts up. Default: 0

5.45.7 Configuring Data Durability

RTI Connext implements **DDS_TRANSIENT_DURABILITY_QOS** (p.1288) and **DDS_PERSISTENT_DURABILITY_**↵ **QOS** (p.1288) 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.1285)

5.46 Basic QoS

QoS that are considered basic and fundamental to RTI Connex.

QoS that are considered basic and fundamental to RTI Connex.

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 Connex, and showcase RTI Connex 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. 1759)	kind (p. 1762)
	<< basic >> (p. 783) << spec >> (p. 790) The kind of durability.
DDS_DurabilityQosPolicy (p. 1759)	writer_depth (p. 1762)
	<< basic >> (p. 783) << extension >> (p. 795) Indicates the number of samples per instance that a durable DDS_DataWriter (p. 485) will send to a late-joining DDS_DataReader (p. 615).
DDS_DataRepresentationQosPolicy (p. 1638)	compression_settings (p. 1640)
	<< basic >> (p. 783) << extension >> (p. 795) Structure that contains the compression settings.
DDS_DeadlineQosPolicy (p. 1684)	period (p. 1685)
	<< basic >> (p. 783) << spec >> (p. 790) Duration of the deadline period.
DDS_OwnershipQosPolicy (p. 1863)	kind (p. 1868)
	<< basic >> (p. 783) << spec >> (p. 790) The kind of ownership
DDS_OwnershipStrengthQosPolicy (p. 1868)	value (p. 1869)
	<< basic >> (p. 783) << spec >> (p. 790) The strength value used to arbitrate among multiple writers.
DDS_LivelinessQosPolicy (p. 1823)	kind (p. 1825)
	<< basic >> (p. 783) << spec >> (p. 790) The kind of liveliness desired.
DDS_LivelinessQosPolicy (p. 1823)	lease_duration (p. 1825)
	<< basic >> (p. 783) << spec >> (p. 790) The duration within which a DDS_DataWriter (p. 485) must be asserted, or else it is assumed to be not alive.

DDS_LivelinessQosPolicy (p. 1823)	assertions_per_lease_duration (p. 1826)
	<< basic >> (p. 783) << extension >> (p. 795) The number of assertions a DDS_DataWriter (p. 485) will send during a its DDS_LivelinessQosPolicy::lease_duration (p. 1825).
DDS_TimeBasedFilterQosPolicy (p. 2032)	minimum_separation (p. 2034)
	<< basic >> (p. 783) << spec >> (p. 790) The minimum separation duration between subsequent samples.
DDS_PartitionQosPolicy (p. 1881)	name (p. 1883)
	<< basic >> (p. 783) << spec >> (p. 790) A list of partition names
DDS_ReliabilityQosPolicy (p. 1936)	kind (p. 1939)
	<< basic >> (p. 783) << spec >> (p. 790) Kind of reliability.
DDS_ReliabilityQosPolicy (p. 1936)	max_blocking_time (p. 1939)
	<< basic >> (p. 783) << spec >> (p. 790) The maximum time a DataWriter can block on a write() call.
DDS_DestinationOrderQosPolicy (p. 1686)	kind (p. 1688)
	<< basic >> (p. 783) << spec >> (p. 790) Specifies the desired kind of destination order.
DDS_HistoryQosPolicy (p. 1805)	kind (p. 1807)
	<< basic >> (p. 783) << spec >> (p. 790) Specifies the kind of history to be kept.
DDS_HistoryQosPolicy (p. 1805)	depth (p. 1807)
	<< basic >> (p. 783) << spec >> (p. 790) Specifies the number of samples per instance to be kept, when the <code>kind</code> is DDS_KEEP_LAST_HISTORY_QOS (p. 1293)
DDS_ResourceLimitsQosPolicy (p. 1949)	max_samples (p. 1951)
	<< basic >> (p. 783) << spec >> (p. 790) Represents the maximum samples the middleware can store for any one DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)).
DDS_ResourceLimitsQosPolicy (p. 1949)	max_instances (p. 1951)
	<< basic >> (p. 783) << spec >> (p. 790) Represents the maximum number of instances a DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)) can manage.
DDS_ResourceLimitsQosPolicy (p. 1949)	initial_samples (p. 1952)
	<< basic >> (p. 783) << extension >> (p. 795) Represents the initial samples the middleware will store for any one DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)).

DDS_ResourceLimitsQosPolicy (p. 1949)	initial_instances (p. 1952)
	<< basic >> (p. 783) << extension >> (p. 795) Represents the initial number of instances a DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)) will manage.
DDS_EntityFactoryQosPolicy (p. 1788)	autoenable_created_entities (p. 1789)
	<< basic >> (p. 783) << spec >> (p. 790) Specifies whether the entity acting as a factory automatically enables the instances it creates.
DDS_LifespanQosPolicy (p. 1815)	duration (p. 1816)
	<< basic >> (p. 783) << spec >> (p. 790) Maximum duration for the data's validity.
DDS_PublishModeQosPolicy (p. 1920)	kind (p. 1922)
	<< basic >> (p. 783) << extension >> (p. 795) Publishing mode
DDS_WireProtocolQosPolicy (p. 2093)	participant_id (p. 2097)
	<< basic >> (p. 783) << extension >> (p. 795) A value used to distinguish among different participants belonging to the same domain on the same host.
DDS_TransportBuiltinQosPolicy (p. 2051)	mask (p. 2052)
	<< basic >> (p. 783) << extension >> (p. 795) Specifies the built-in transports that are registered automatically when the DDS_DomainParticipant (p. 77) is enabled.
DDS_TransportMulticastQosPolicy (p. 2058)	value (p. 2059)
	<< basic >> (p. 783) << extension >> (p. 795) A sequence of multicast communications settings.
DDS_DiscoveryQosPolicy (p. 1715)	initial_peers (p. 1716)
	<< basic >> (p. 783) << extension >> (p. 795) 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. 1715)	multicast_receive_addresses (p. 1717)
	<< basic >> (p. 783) << extension >> (p. 795) Specifies the multicast group addresses on which discovery-related meta-traffic can be received by the DomainParticipant.
DDS_DiscoveryQosPolicy (p. 1715)	accept_unknown_peers (p. 1718)

	<< basic >> (p. 783) << extension >> (p. 795) Whether to accept a new participant that is not in the initial peers list.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_liveliness_lease_duration (p. 1692)
	<< basic >> (p. 783) << extension >> (p. 795) The liveliness lease duration for the participant.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_liveliness_assert_period (p. 1692)
	<< basic >> (p. 783) << extension >> (p. 795) The period to assert liveliness for the participant.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_announcement_period (p. 1693)
	<< basic >> (p. 783) << extension >> (p. 795) The period at which a participant announces itself to potential peers when using the Simple Participant Discovery Protocol 2.0 (SPDP2).
DDS_DiscoveryConfigQosPolicy (p. 1689)	initial_participant_announcements (p. 1694)
	<< basic >> (p. 783) << extension >> (p. 795) The number of initial announcements sent when a participant is first enabled.
DDS_DiscoveryConfigQosPolicy (p. 1689)	new_remote_participant_announcements (p. 1695)
	<< basic >> (p. 783) << extension >> (p. 795) The number of participant announcements sent when a remote participant is newly discovered.
DDS_DiscoveryConfigQosPolicy (p. 1689)	min_initial_participant_announcement_period (p. 1695)
	<< basic >> (p. 783) << extension >> (p. 795) The minimum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
DDS_DiscoveryConfigQosPolicy (p. 1689)	max_initial_participant_announcement_period (p. 1695)
	<< basic >> (p. 783) << extension >> (p. 795) The maximum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	heartbeat_period (p. 1963)
	<< basic >> (p. 783) << extension >> (p. 795) The period at which to send heartbeats.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	fast_heartbeat_period (p. 1963)
	<< basic >> (p. 783) << extension >> (p. 795) An alternative heartbeat period used when a reliable writer needs to flush its unacknowledged samples more quickly.

DDS_RtpsReliableWriterProtocol_t (p. 1960)	late_joiner_heartbeat_period (p. 1964)
	<< basic >> (p. 783) << extension >> (p. 795) 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. 1960)	min_send_window_size (p. 1973)
	<< basic >> (p. 783) << extension >> (p. 795) Minimum size of send window of unacknowledged samples
DDS_RtpsReliableWriterProtocol_t (p. 1960)	max_send_window_size (p. 1973)
	<< basic >> (p. 783) << extension >> (p. 795) Maximum size of send window of unacknowledged samples
DDS_PropertyQosPolicy (p. 1901)	value (p. 1903)
	<< basic >> (p. 783) << extension >> (p. 795) Sequence of properties.
DDS_EntityNameQosPolicy (p. 1790)	name (p. 1790)
	<< basic >> (p. 783) << extension >> (p. 795) The name of the entity.
DDS_BatchQosPolicy (p. 1552)	enable (p. 1552)
	<< basic >> (p. 783) << extension >> (p. 795) Specifies whether or not batching is enabled.
DDS_BatchQosPolicy (p. 1552)	max_data_bytes (p. 1553)
	<< basic >> (p. 783) << extension >> (p. 795) The maximum cumulative length of all serialized samples in a batch.
DDS_BatchQosPolicy (p. 1552)	max_samples (p. 1553)
	<< basic >> (p. 783) << extension >> (p. 795) The maximum number of samples in a batch.
DDS_ProfileQosPolicy (p. 1898)	string_profile (p. 1899)
	<< basic >> (p. 783) << extension >> (p. 795) Sequence of strings containing a XML document to load.
DDS_ProfileQosPolicy (p. 1898)	url_profile (p. 1899)
	<< basic >> (p. 783) << extension >> (p. 795) Sequence of URL groups (p. 973) containing a set of XML documents to load.
DDS_ProfileQosPolicy (p. 1898)	ignore_user_profile (p. 1899)
	<< basic >> (p. 783) << extension >> (p. 795) Ignores the file USER_QOS_PROFILES.xml in the current working directory.

DDS_ProfileQosPolicy (p. 1898)	ignore_environment_profile (p. 1899)
	<< basic >> (p. 783) << extension >> (p. 795) Ignores the value of the NDDS_QOS_PROFILES (p. 973).
DDS_ProfileQosPolicy (p. 1898)	ignore_resource_profile (p. 1900)
	<< basic >> (p. 783) << extension >> (p. 795) Ignores the file NDDS_QOS_PROFILES.xml .
DDS_LoggingQosPolicy (p. 1831)	verbosity (p. 1832)
	<< basic >> (p. 783) << extension >> (p. 795) The verbosity at which RTI Connex diagnostic information is logged.
DDS_LoggingQosPolicy (p. 1831)	category (p. 1832)
	<< basic >> (p. 783) << extension >> (p. 795) Categories of logged messages.
DDS_LoggingQosPolicy (p. 1831)	output_file (p. 1832)
	<< basic >> (p. 783) << extension >> (p. 795) Specifies the file to which log messages will be redirected to.
DDS_LoggingQosPolicy (p. 1831)	output_file_suffix (p. 1833)
	<< basic >> (p. 783) << extension >> (p. 795) Sets the file suffix when logging to a set of files
DDS_MonitoringEventDistributionSettings (p. 1840)	publication_period (p. 1843)
	<< basic >> (p. 783) << extension >> (p. 795) Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.
DDS_MonitoringPeriodicDistributionSettings (p. 1852)	polling_period (p. 1853)
	<< basic >> (p. 783) << extension >> (p. 795) Period at which the periodic metric thread polls and publishes the periodic metrics.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	max_historical_logs (p. 1845)
	<< basic >> (p. 783) << extension >> (p. 795) The number of log messages that RTI Monitoring Library 2.0 will keep as history.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	publication_period (p. 1846)
	<< basic >> (p. 783) << extension >> (p. 795) Period at which the logging thread publishes log messages.

DDS_MonitoringDedicatedParticipantSettings (p. 1836)	enable (p. 1836)
	<< basic >> (p. 783) << extension >> (p. 795) Enables the use of a dedicated DDS_DomainParticipant (p. 77) to distribute the RTI Connex application telemetry data.
DDS_MonitoringDedicatedParticipantSettings (p. 1836)	domain_id (p. 1837)
	<< basic >> (p. 783) << extension >> (p. 795) The domain ID used in the creation of RTI Monitoring Library 2.0 DDS_DomainParticipant (p. 77).
DDS_MonitoringDedicatedParticipantSettings (p. 1836)	collector_initial_peers (p. 1837)
	<< basic >> (p. 783) << extension >> (p. 795) 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. 1839).
DDS_MonitoringLoggingForwardingSettings (p. 1847)	middleware_forwarding_level (p. 1847)
	<< basic >> (p. 783) << extension >> (p. 795) Log messages with NDDS_CONFIG_LOG_FACILITY ↵ MIDDLEWARE (p. 1447) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1847)	security_event_forwarding_level (p. 1847)
	<< basic >> (p. 783) << extension >> (p. 795) Log messages with NDDS_CONFIG_LOG_FACILITY ↵ SECURITY_EVENT (p. 1447) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1847)	service_forwarding_level (p. 1848)
	<< basic >> (p. 783) << extension >> (p. 795) Log messages with NDDS_CONFIG_LOG_FACILITY ↵ SERVICE (p. 1447) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1847)	user_forwarding_level (p. 1848)
	<< basic >> (p. 783) << extension >> (p. 795) Log messages with NDDS_CONFIG_LOG_FACILITY ↵ USER (p. 1447) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.

DDS_MonitoringTelemetryData (p. 1855)	metrics (p. 1855)
	<< basic >> (p. 783) << extension >> (p. 795) Sequence of DDS_MonitoringMetricSelection (p. 1849) containing the event and periodic metrics that will be collected and distributed for a given set of observable resources.
DDS_MonitoringTelemetryData (p. 1855)	logs (p. 1856)
	<< basic >> (p. 783) << extension >> (p. 795) DDS_MonitoringLoggingForwardingSettings (p. 1847) containing the NDDS_Config_Syslog ↔ Verbosity (p. 1447) levels that will be forwarded for the different NDDS_Config_LogFacility (p. 1446).
DDS_MonitoringQosPolicy (p. 1853)	enable (p. 1854)
	<< basic >> (p. 783) << extension >> (p. 795) Enables the collection and distribution of telemetry data for an RTI Connex application using RTI Monitoring Library 2.0.
DDS_MonitoringQosPolicy (p. 1853)	application_name (p. 1854)
	<< basic >> (p. 783) << extension >> (p. 795) The name of the resource that represents this RTI Connex application.
DDS_TopicQueryDispatchQosPolicy (p. 2047)	publication_period (p. 2048)
	<< basic >> (p. 783) << extension >> (p. 795) Sets the periodic interval at which samples are published.

5.47 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.dds-foundation.org/omg-dds-standard/>.

DDS_UserDataQosPolicy (p. 2086)	value (p. 2087)
	<< spec >> (p. 790) a sequence of octets
DDS_GroupDataQosPolicy (p. 1803)	value (p. 1804)
	<< spec >> (p. 790) a sequence of octets
DDS_DurabilityQosPolicy (p. 1759)	kind (p. 1762)

	<< basic >> (p. 783) << spec >> (p. 790) The kind of durability.
DDS_DurabilityServiceQosPolicy (p. 1764)	service_cleanup_delay (p. 1765)
	<< spec >> (p. 790) Controls when the service is able to remove all information regarding a data instances.
DDS_DurabilityServiceQosPolicy (p. 1764)	history_kind (p. 1765)
	<< spec >> (p. 790) The kind of history to apply in re-couping durable data.
DDS_DurabilityServiceQosPolicy (p. 1764)	history_depth (p. 1765)
	<< spec >> (p. 790) Setting to use for the DDS_↔DurabilityQosPolicy::writer_depth (p. 1762) when re-couping durable data.
DDS_DurabilityServiceQosPolicy (p. 1764)	max_samples (p. 1766)
	<< spec >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1764)	max_instances (p. 1766)
	<< spec >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1764)	max_samples_per_instance (p. 1766)
	<< spec >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DataRepresentationQosPolicy (p. 1638)	value (p. 1640)
	<< spec >> (p. 790) Sequence of representation identifiers.
DDS_PresentationQosPolicy (p. 1890)	access_scope (p. 1893)
	<< spec >> (p. 790) Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
DDS_PresentationQosPolicy (p. 1890)	coherent_access (p. 1893)
	<< spec >> (p. 790) Specifies support for <i>coherent</i> access. Controls whether coherent access is supported within the scope <code>access_scope</code> .
DDS_PresentationQosPolicy (p. 1890)	ordered_access (p. 1893)
	<< spec >> (p. 790) Specifies support for <i>ordered</i> access to the samples received at the subscription end. Controls whether ordered access is supported within the scope <code>access_scope</code> .
DDS_DeadlineQosPolicy (p. 1684)	period (p. 1685)

	<< basic >> (p. 783) << spec >> (p. 790) Duration of the deadline period.
DDS_LatencyBudgetQosPolicy (p. 1813)	duration (p. 1815)
	<< spec >> (p. 790) Duration of the maximum acceptable delay.
DDS_OwnershipQosPolicy (p. 1863)	kind (p. 1868)
	<< basic >> (p. 783) << spec >> (p. 790) The kind of ownership
DDS_OwnershipStrengthQosPolicy (p. 1868)	value (p. 1869)
	<< basic >> (p. 783) << spec >> (p. 790) The strength value used to arbitrate among multiple writers.
DDS_LivelinessQosPolicy (p. 1823)	kind (p. 1825)
	<< basic >> (p. 783) << spec >> (p. 790) The kind of liveliness desired.
DDS_LivelinessQosPolicy (p. 1823)	lease_duration (p. 1825)
	<< basic >> (p. 783) << spec >> (p. 790) The duration within which a DDS_DataWriter (p. 485) must be asserted, or else it is assumed to be not alive.
DDS_TimeBasedFilterQosPolicy (p. 2032)	minimum_separation (p. 2034)
	<< basic >> (p. 783) << spec >> (p. 790) The minimum separation duration between subsequent samples.
DDS_PartitionQosPolicy (p. 1881)	name (p. 1883)
	<< basic >> (p. 783) << spec >> (p. 790) A list of partition names
DDS_ReliabilityQosPolicy (p. 1936)	kind (p. 1939)
	<< basic >> (p. 783) << spec >> (p. 790) Kind of reliability.
DDS_ReliabilityQosPolicy (p. 1936)	max_blocking_time (p. 1939)
	<< basic >> (p. 783) << spec >> (p. 790) The maximum time a DataWriter can block on a write() call.
DDS_DestinationOrderQosPolicy (p. 1686)	kind (p. 1688)
	<< basic >> (p. 783) << spec >> (p. 790) Specifies the desired kind of destination order.
DDS_DestinationOrderQosPolicy (p. 1686)	scope (p. 1688)
	<< spec >> (p. 790) Specifies the desired scope of the source destination order.
DDS_HistoryQosPolicy (p. 1805)	kind (p. 1807)

	<< basic >> (p. 783) << spec >> (p. 790) Specifies the kind of history to be kept.
DDS_HistoryQosPolicy (p. 1805)	depth (p. 1807)
	<< basic >> (p. 783) << spec >> (p. 790) Specifies the number of samples per instance to be kept, when the kind is DDS_KEEP_LAST_HISTORY_QOS (p. 1293)
DDS_ResourceLimitsQosPolicy (p. 1949)	max_samples (p. 1951)
	<< basic >> (p. 783) << spec >> (p. 790) Represents the maximum samples the middleware can store for any one DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)).
DDS_ResourceLimitsQosPolicy (p. 1949)	max_instances (p. 1951)
	<< basic >> (p. 783) << spec >> (p. 790) Represents the maximum number of instances a DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)) can manage.
DDS_ResourceLimitsQosPolicy (p. 1949)	max_samples_per_instance (p. 1952)
	<< spec >> (p. 790) Represents the maximum number of samples of any one instance a DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)) can manage.
DDS_EntityFactoryQosPolicy (p. 1788)	autoenable_created_entities (p. 1789)
	<< basic >> (p. 783) << spec >> (p. 790) Specifies whether the entity acting as a factory automatically enables the instances it creates.
DDS_TransportPriorityQosPolicy (p. 2062)	value (p. 2064)
	<< spec >> (p. 790) 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. 1815)	duration (p. 1816)
	<< basic >> (p. 783) << spec >> (p. 790) Maximum duration for the data's validity.
DDS_WriterDataLifecycleQosPolicy (p. 2106)	autodispose_unregistered_instances (p. 2107)
	<< spec >> (p. 790) Boolean flag that controls whether or not a writer will also dispose of an instance when the DDS_DataWriter (p. 485) unregisters from the instance by means of the unregister operations.
DDS_ReaderDataLifecycleQosPolicy (p. 1930)	autopurge_nowriter_samples_delay (p. 1932)
	<< spec >> (p. 790) Minimum duration for which the DDS_DataReader (p. 615) will maintain samples for an instance once its <code>instance_state</code> becomes DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE (p. 715).

DDS_ReaderDataLifecycleQosPolicy (p. 1930)	autopurge_disposed_samples_delay (p. 1932)
	<< <i>spec</i> >> (p. 790) Minimum duration for which the DDS_DataReader (p. 615) will maintain samples for an instance once its <code>instance_state</code> becomes DDS_NOT_ALIVE_DISPOSED_INSTANCE_↔STATE (p. 715).
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	kind (p. 2077)
	<< <i>spec</i> >> (p. 790) Type consistency kind.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	ignore_sequence_bounds (p. 2077)
	<< <i>spec</i> >> (p. 790) Controls whether sequence bounds are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	ignore_string_bounds (p. 2078)
	<< <i>spec</i> >> (p. 790) Controls whether string bounds are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	ignore_member_names (p. 2078)
	<< <i>spec</i> >> (p. 790) Controls whether member names are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	prevent_type_widening (p. 2078)
	<< <i>spec</i> >> (p. 790) Controls whether type widening is allowed.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	force_type_validation (p. 2079)
	<< <i>spec</i> >> (p. 790) Controls whether type information must be available in order to complete matching between a DDS_DataWriter (p. 485) and a DDS_Data↔Reader (p. 615).
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	ignore_enum_literal_names (p. 2079)
	<< <i>spec</i> >> (p. 790) Controls whether enumeration constant names are taken into consideration for type assignability.
DDS_DataTags (p. 1641)	tags (p. 1641)
	<< <i>spec</i> >> (p. 790) Sequence of data tags.

5.48 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. 1547)	enable_required_subscriptions (p. 1550)
	<< extension >> (p. 795) Enables support for required subscriptions in a DDS_DataWriter (p. 485)
DDS_AvailabilityQosPolicy (p. 1547)	max_data_availability_waiting_time (p. 1550)
	<< extension >> (p. 795) Defines how much time to wait before delivering a sample to the application without having received some of the previous samples.
DDS_AvailabilityQosPolicy (p. 1547)	max_endpoint_availability_waiting_time (p. 1550)
	<< extension >> (p. 795) Defines how much time to wait to discover DataWriters providing samples for the same data source (virtual GUID).
DDS_AvailabilityQosPolicy (p. 1547)	required_matched_endpoint_groups (p. 1551)
	<< extension >> (p. 795) A sequence of endpoint groups.
DDS_DurabilityQosPolicy (p. 1759)	direct_communication (p. 1762)
	<< extension >> (p. 795) Indicates whether or not a TRANSIENT or PERSISTENT DDS_DataReader (p. 615) should receive samples directly from a TRANSIENT or PERSISTENT DDS_DataWriter (p. 485)
DDS_DurabilityQosPolicy (p. 1759)	writer_depth (p. 1762)
	<< basic >> (p. 783) << extension >> (p. 795) Indicates the number of samples per instance that a durable DDS_DataWriter (p. 485) will send to a late-joining DDS_DataReader (p. 615).
DDS_DurabilityQosPolicy (p. 1759)	storage_settings (p. 1763)
	<< extension >> (p. 795) Configures durable writer history and durable reader state.
DDS_PersistentStorageSettings (p. 1883)	enable (p. 1885)
	<< extension >> (p. 795) Enables durable writer history in a DDS_DataWriter (p. 485) and durable reader state in a DDS_DataReader (p. 615).

DDS_PersistentStorageSettings (p. 1883)	file_name (p. 1885)
	<< <i>extension</i> >> (p. 795) The file name where the durable writer history or durable reader state will be stored.
DDS_PersistentStorageSettings (p. 1883)	trace_file_name (p. 1885)
	<< <i>extension</i> >> (p. 795) 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. 1883)	journal_kind (p. 1886)
	<< <i>extension</i> >> (p. 795) Sets the journal mode of the persistent storage.
DDS_PersistentStorageSettings (p. 1883)	synchronization_kind (p. 1886)
	<< <i>extension</i> >> (p. 795) Sets the level of synchronization with the physical disk.
DDS_PersistentStorageSettings (p. 1883)	vacuum (p. 1886)
	<< <i>extension</i> >> (p. 795) Sets the auto-vacuum status of the storage.
DDS_PersistentStorageSettings (p. 1883)	restore (p. 1887)
	<< <i>extension</i> >> (p. 795) Indicates if the persisted writer history or reader state must be restored.
DDS_PersistentStorageSettings (p. 1883)	writer_instance_cache_allocation (p. 1887)
	<< <i>extension</i> >> (p. 795) Configures the resource limits associated with the instance durable writer history cache.
DDS_PersistentStorageSettings (p. 1883)	writer_sample_cache_allocation (p. 1888)
	<< <i>extension</i> >> (p. 795) Configures the resource limits associated with the sample durable writer history cache.
DDS_PersistentStorageSettings (p. 1883)	writer_memory_state (p. 1889)
	<< <i>extension</i> >> (p. 795) 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. 1883)	reader_checkpoint_frequency (p. 1889)
	<< <i>extension</i> >> (p. 795) Controls how often the reader state is stored into the database.
DDS_CompressionSettings_t (p. 1568)	compression_ids (p. 1568)
	<< <i>extension</i> >> (p. 795) Mask that represents the compression algorithms enabled.
DDS_CompressionSettings_t (p. 1568)	writer_compression_level (p. 1569)

	<< extension >> (p. 795) The level of compression to use when compressing data.
DDS_CompressionSettings_t (p. 1568)	writer_compression_threshold (p. 1569)
	<< extension >> (p. 795) The threshold, in bytes, above which a serialized sample will be eligible to be compressed.
DDS_DataRepresentationQosPolicy (p. 1638)	compression_settings (p. 1640)
	<< basic >> (p. 783) << extension >> (p. 795) Structure that contains the compression settings.
DDS_PresentationQosPolicy (p. 1890)	drop_incomplete_coherent_set (p. 1894)
	<< extension >> (p. 795) Indicates whether or not a DDS_DataReader (p. 615) 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. 1823)	assertions_per_lease_duration (p. 1826)
	<< basic >> (p. 783) << extension >> (p. 795) The number of assertions a DDS_DataWriter (p. 485) will send during a its DDS_LivelinessQosPolicy::lease_↔duration (p. 1825).
DDS_ReliabilityQosPolicy (p. 1936)	acknowledgment_kind (p. 1939)
	<< extension >> (p. 795) Kind of reliable acknowledgment
DDS_ReliabilityQosPolicy (p. 1936)	instance_state_consistency_kind (p. 1940)
	<< extension >> (p. 795) Whether instance state consistency is enabled
DDS_DestinationOrderQosPolicy (p. 1686)	source_timestamp_tolerance (p. 1688)
	<< extension >> (p. 795) Allowed tolerance between source timestamps of consecutive samples.
DDS_ResourceLimitsQosPolicy (p. 1949)	initial_samples (p. 1952)
	<< basic >> (p. 783) << extension >> (p. 795) Represents the initial samples the middleware will store for any one DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)).
DDS_ResourceLimitsQosPolicy (p. 1949)	initial_instances (p. 1952)
	<< basic >> (p. 783) << extension >> (p. 795) Represents the initial number of instances a DDS_Data↔Writer (p. 485) (or DDS_DataReader (p. 615)) will manage.
DDS_ResourceLimitsQosPolicy (p. 1949)	instance_hash_buckets (p. 1953)

	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for instances.
DDS_TypeSupportQosPolicy (p. 2081)	plugin_data (p. 2082)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Value to pass into the type plugin's de-/serialization function
DDS_TypeSupportQosPolicy (p. 2081)	cdr_padding_kind (p. 2082)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Determines whether or not the padding bytes will be set to zero during CDR serialization.
DDS_PublishModeQosPolicy (p. 1920)	kind (p. 1922)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) Publishing mode
DDS_PublishModeQosPolicy (p. 1920)	flow_controller_name (p. 1922)
	<< <i>extension</i> >> (p. 795) Name of the associated flow controller
DDS_PublishModeQosPolicy (p. 1920)	priority (p. 1923)
	<< <i>extension</i> >> (p. 795) Publication priority
DDS_AsynchronousPublisherQosPolicy (p. 1539)	disable_asynchronous_write (p. 1540)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Disable asynchronous publishing.
DDS_AsynchronousPublisherQosPolicy (p. 1539)	thread (p. 1540)
	<< <i>extension</i> >> (p. 795) Settings of the publishing thread.
DDS_AsynchronousPublisherQosPolicy (p. 1539)	disable_asynchronous_batch (p. 1541)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Disable asynchronous batch flushing.
DDS_AsynchronousPublisherQosPolicy (p. 1539)	asynchronous_batch_thread (p. 1541)
	<< <i>extension</i> >> (p. 795) Settings of the batch flushing thread.
DDS_AsynchronousPublisherQosPolicy (p. 1539)	disable_topic_query_publication (p. 1542)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Disable topic query publication.
DDS_AsynchronousPublisherQosPolicy (p. 1539)	topic_query_publication_thread (p. 1542)
	<< <i>extension</i> >> (p. 795) Settings of the DDS_Topic↔Query (p. 707) publication thread.
DDS_WriterDataLifecycleQosPolicy (p. 2106)	autopurge_unregistered_instances_delay (p. 2107)

	<< extension >> (p. 795) Maximum duration for which the DDS_DataWriter (p. 485) will maintain information regarding an instance once it has unregistered the instance.
DDS_WriterDataLifecycleQosPolicy (p. 2106)	autopurge_disposed_instances_delay (p. 2108)
	<< extension >> (p. 795) Maximum duration for which the DDS_DataWriter (p. 485) will maintain information regarding an instance once it has disposed the instance.
DDS_ReaderDataLifecycleQosPolicy (p. 1930)	autopurge_disposed_instances_delay (p. 1932)
	<< extension >> (p. 795) Minimum duration for which the DDS_DataReader (p. 615) will maintain information about a received instance once its <code>instance_state</code> becomes DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE (p. 715) and there are no samples for the instance in the DataReader queue.
DDS_ReaderDataLifecycleQosPolicy (p. 1930)	autopurge_nowriter_instances_delay (p. 1933)
	<< extension >> (p. 795) Minimum duration for which the DDS_DataReader (p. 615) will maintain information about a received instance once its <code>instance_state</code> becomes DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE (p. 715) and there are no samples for the instance in the DataReader queue.
DDS_WireProtocolQosPolicy (p. 2093)	participant_id (p. 2097)
	<< basic >> (p. 783) << extension >> (p. 795) A value used to distinguish among different participants belonging to the same domain on the same host.
DDS_WireProtocolQosPolicy (p. 2093)	rtps_host_id (p. 2097)
	<< extension >> (p. 795) The RTPS Host ID of the domain participant.
DDS_WireProtocolQosPolicy (p. 2093)	rtps_app_id (p. 2098)
	<< extension >> (p. 795) The RTPS App ID of the domain participant.
DDS_WireProtocolQosPolicy (p. 2093)	rtps_instance_id (p. 2098)
	<< extension >> (p. 795) The RTPS Instance ID of the DDS_DomainParticipant (p. 77).
DDS_WireProtocolQosPolicy (p. 2093)	rtps_well_known_ports (p. 2099)
	<< extension >> (p. 795) Configures the RTPS well-known port mappings.
DDS_WireProtocolQosPolicy (p. 2093)	rtps_reserved_port_mask (p. 2099)
	<< extension >> (p. 795) Specifies which well-known ports to reserve when enabling the participant.

DDS_WireProtocolQosPolicy (p. 2093)	rtps_auto_id_kind (p. 2100)
	<< extension >> (p. 795) Kind of auto mechanism used to calculate the GUID prefix.
DDS_WireProtocolQosPolicy (p. 2093)	compute_crc (p. 2100)
	<< extension >> (p. 795) Adds RTPS CRC submessage to every message when this field is set to DDS_↵ BOOLEAN_TRUE (p. 1201).
DDS_WireProtocolQosPolicy (p. 2093)	check_crc (p. 2100)
	<< extension >> (p. 795) 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. 1201).
DDS_TransportSelectionQosPolicy (p. 2064)	enabled_transports (p. 2065)
	<< extension >> (p. 795) A sequence of transport aliases that specifies the transport instances available for use by the entity.
DDS_TransportBuiltinQosPolicy (p. 2051)	mask (p. 2052)
	<< basic >> (p. 783) << extension >> (p. 795) Specifies the built-in transports that are registered automatically when the DDS_DomainParticipant (p. 77) is enabled.
DDS_TransportUnicastQosPolicy (p. 2065)	value (p. 2067)
	<< extension >> (p. 795) A sequence of unicast communication settings.
DDS_TransportMulticastMappingQosPolicy (p. 2056)	value (p. 2058)
	<< extension >> (p. 795) A sequence of multicast communication mappings.
DDS_TransportMulticastQosPolicy (p. 2058)	value (p. 2059)
	<< basic >> (p. 783) << extension >> (p. 795) A sequence of multicast communications settings.
DDS_TransportMulticastQosPolicy (p. 2058)	kind (p. 2059)
	<< extension >> (p. 795) A value that specifies a way to determine how to obtain the multicast address.
DDS_DiscoveryQosPolicy (p. 1715)	enabled_transports (p. 1716)
	<< extension >> (p. 795) The transports available for use by the Discovery mechanism.
DDS_DiscoveryQosPolicy (p. 1715)	initial_peers (p. 1716)

	<< basic >> (p. 783) << extension >> (p. 795) 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. 1715)	multicast_receive_addresses (p. 1717)
	<< basic >> (p. 783) << extension >> (p. 795) Specifies the multicast group addresses on which discovery-related meta-traffic can be received by the Domain↔ Participant.
DDS_DiscoveryQosPolicy (p. 1715)	metatraffic_transport_priority (p. 1717)
	<< extension >> (p. 795) The transport priority to use for the Discovery meta-traffic.
DDS_DiscoveryQosPolicy (p. 1715)	accept_unknown_peers (p. 1718)
	<< basic >> (p. 783) << extension >> (p. 795) Whether to accept a new participant that is not in the initial peers list.
DDS_DiscoveryQosPolicy (p. 1715)	enable_endpoint_discovery (p. 1718)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Whether to automatically enable endpoint discovery for all the remote participants.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_remote_writers (p. 1621)
	<< extension >> (p. 795) The maximum number of remote writers from which a DDS_DataReader (p. 615) may read, including all instances.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_remote_writers_per_instance (p. 1622)
	<< extension >> (p. 795) The maximum number of remote writers from which a DDS_DataReader (p. 615) may read a single instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_samples_per_remote_writer (p. 1622)
	<< extension >> (p. 795) The maximum number of out-of-order samples from a given remote DDS_DataWriter (p. 485) that a DDS_DataReader (p. 615) may store when maintaining a reliable connection to the DDS_↔DataWriter (p. 485).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_infos (p. 1623)
	<< extension >> (p. 795) The maximum number of info units that a DDS_DataReader (p. 615) can use to store DDS_SampleInfo (p. 1985).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_remote_writers (p. 1623)
	<< extension >> (p. 795) The initial number of remote writers from which a DDS_DataReader (p. 615) may read, including all instances.

DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_remote_writers_per_instance (p. 1624)
	<< <i>extension</i> >> (p. 795) The initial number of remote writers from which a DDS_DataReader (p. 615) may read a single instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_infos (p. 1624)
	<< <i>extension</i> >> (p. 795) The initial number of info units that a DDS_DataReader (p. 615) can have, which are used to store DDS_SampleInfo (p. 1985).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_outstanding_reads (p. 1625)
	<< <i>extension</i> >> (p. 795) The initial number of outstanding calls to read/take (or one of their variants) on the same DDS_DataReader (p. 615) for which memory has not been returned by calling FooDataReader_return_loan (p. 646).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_outstanding_reads (p. 1625)
	<< <i>extension</i> >> (p. 795) The maximum number of outstanding read/take calls (or one of their variants) on the same DDS_DataReader (p. 615) for which memory has not been returned by calling FooDataReader_return_loan (p. 646).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_samples_per_read (p. 1625)
	<< <i>extension</i> >> (p. 795) The maximum number of data samples that the application can receive from the middleware in a single call to FooDataReader_read (p. 625) or FooDataReader_take (p. 626). If more data exists in the middleware, the application will need to issue multiple read/take calls.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	disable_fragmentation_support (p. 1626)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Determines whether the DDS_DataReader (p. 615) can receive fragmented samples.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_fragmented_samples (p. 1626)
	<< <i>extension</i> >> (p. 795) The maximum number of samples for which the DDS_DataReader (p. 615) may store fragments at a given point in time.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_fragmented_samples (p. 1627)
	<< <i>extension</i> >> (p. 795) The initial number of samples for which a DDS_DataReader (p. 615) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_fragmented_samples_per_remote_writer (p. 1627)

	<< <i>extension</i> >> (p. 795) The maximum number of samples per remote writer for which a DDS_DataReader (p. 615) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_fragments_per_sample (p. 1628)
	<< <i>extension</i> >> (p. 795) Maximum number of fragments for a single sample.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	dynamically_allocate_fragmented_samples (p. 1628)
	<< <i>extension</i> >> (p. 795) Determines whether the DDS_DataReader (p. 615) pre-allocates storage for storing fragmented samples.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_total_instances (p. 1629)
	<< <i>extension</i> >> (p. 795) Maximum number of instances for which a DataReader will keep state.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_remote_virtual_writers (p. 1630)
	<< <i>extension</i> >> (p. 795) The maximum number of remote virtual writers from which a DDS_DataReader (p. 615) may read, including all instances.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_remote_virtual_writers (p. 1630)
	<< <i>extension</i> >> (p. 795) The initial number of remote virtual writers from which a DDS_DataReader (p. 615) may read, including all instances.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_remote_virtual_writers_per_instance (p. 1631)
	<< <i>extension</i> >> (p. 795) The maximum number of virtual remote writers that can be associated with an instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_remote_virtual_writers_per_instance (p. 1631)
	<< <i>extension</i> >> (p. 795) The initial number of virtual remote writers per instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_remote_writers_per_sample (p. 1632)
	<< <i>extension</i> >> (p. 795) The maximum number of remote writers allowed to write the same sample.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_query_condition_filters (p. 1632)
	<< <i>extension</i> >> (p. 795) The maximum number of query condition filters a reader is allowed.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_app_ack_response_length (p. 1633)
	<< <i>extension</i> >> (p. 795) Maximum length of application-level acknowledgment response data
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	keep_minimum_state_for_instances (p. 1633)

	<< <i>extension</i> >> (p. 795) Whether or not keep a minimum instance state for up to DDS_DataReaderResourceLimitsQosPolicy::max_total_instances (p. 1629).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_topic_queries (p. 1634)
	<< <i>extension</i> >> (p. 795) The initial number of TopicQueries allocated by a DDS_DataReader (p. 615).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_topic_queries (p. 1634)
	<< <i>extension</i> >> (p. 795) The maximum number of active TopicQueries that a DDS_DataReader (p. 615) can create.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	shmem_ref_transfer_mode_attached_segment_allocation (p. 1635)
	<< <i>extension</i> >> (p. 795) Allocation resource for the shared memory segments attached by the DDS_DataReader (p. 615).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	instance_replacement (p. 1635)
	<< <i>extension</i> >> (p. 795) Sets the kind of instances allowed to be replaced for each instance state (DDS_InstanceStateKind (p. 714)) when a DataReader reaches DDS_ResourceLimitsQosPolicy::max_instances (p. 1951).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	autopurge_remote_not_alive_writer_delay (p. 1636)
	<< <i>extension</i> >> (p. 795) Maximum duration for which the DDS_DataReader (p. 615) will maintain information regarding a DDS_DataWriter (p. 485) once the DDS_DataWriter (p. 485) has become not alive.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	initial_concurrent_blocking_threads (p. 1675)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The initial number of threads that are allowed to concurrently block on write call on the same DDS_DataWriter (p. 485).
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_concurrent_blocking_threads (p. 1675)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The maximum number of threads that are allowed to concurrently block on write call on the same DDS_DataWriter (p. 485).
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_remote_reader_filters (p. 1676)
	<< <i>extension</i> >> (p. 795) The maximum number of remote DataReaders for which the DDS_DataWriter (p. 485) will perform content-based filtering.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	initial_batches (p. 1676)

	<< <i>extension</i> >> (p. 795) Represents the initial number of batches a DDS_DataWriter (p. 485) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_batches (p. 1677)
	<< <i>extension</i> >> (p. 795) Represents the maximum number of batches a DDS_DataWriter (p. 485) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	instance_replacement (p. 1677)
	<< <i>extension</i> >> (p. 795) Sets the kinds of instances allowed to be replaced when instance resource limits are reached.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	replace_empty_instances (p. 1678)
	<< <i>extension</i> >> (p. 795) Whether or not to replace empty instances during instance replacement
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	autoregister_instances (p. 1678)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Whether or not to automatically register new instances
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_virtual_writers (p. 1679)
	<< <i>extension</i> >> (p. 795) The maximum number of virtual writers supported by a DDS_DataWriter (p. 485)
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	initial_virtual_writers (p. 1679)
	<< <i>extension</i> >> (p. 795) The initial number of virtual writers supported by a DDS_DataWriter (p. 485)
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_remote_readers (p. 1679)
	<< <i>extension</i> >> (p. 795) The maximum number of remote readers supported by a DDS_DataWriter (p. 485)
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_app_ack_remote_readers (p. 1680)
	<< <i>extension</i> >> (p. 795) The maximum number of application-level acknowledging remote readers supported by a DDS_DataWriter (p. 485)
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	initial_active_topic_queries (p. 1680)
	<< <i>extension</i> >> (p. 795) Represents the initial number of active topic queries a DDS_DataWriter (p. 485) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_active_topic_queries (p. 1680)
	<< <i>extension</i> >> (p. 795) Represents the maximum number of active topic queries a DDS_DataWriter (p. 485) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	writer_loaned_sample_allocation (p. 1681)

	<< extension >> (p. 795) Represents the allocation settings of loaned samples managed by a DDS_DataWriter (p. 485).
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	initialize_writer_loaned_sample (p. 1681)
	<< extension >> (p. 795) Whether or not to initialize loaned samples returned by a DDS_DataWriter (p. 485)
DDS_DataWriterTransferModeQosPolicy (p. 1683)	shmem_ref_settings (p. 1683)
	<< extension >> (p. 795) Settings related to transferring data using shared memory references.
DDS_DataReaderProtocolQosPolicy (p. 1596)	virtual_guid (p. 1597)
	<< extension >> (p. 795) The virtual GUID (Global Unique Identifier).
DDS_DataReaderProtocolQosPolicy (p. 1596)	rtps_object_id (p. 1597)
	<< extension >> (p. 795) The RTPS Object ID.
DDS_DataReaderProtocolQosPolicy (p. 1596)	rtps_reliable_reader (p. 1599)
	<< extension >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a DDS_DataReader (p. 615). This parameter only has effect if the reader is configured with DDS_RELIABLE_RELIABILITY_QOS (p. 1323) DDS_ReliabilityQosPolicyKind (p. 1322).
DDS_DataReaderProtocolQosPolicy (p. 1596)	expects_inline_qos (p. 1597)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Specifies whether this DataReader expects inline QoS with every sample.
DDS_DataReaderProtocolQosPolicy (p. 1596)	disable_positive_acks (p. 1598)
	<< extension >> (p. 795) Whether the reader sends positive acknowledgements to writers.
DDS_DataReaderProtocolQosPolicy (p. 1596)	propagate_dispose_of_unregistered_instances (p. 1598)
	<< extension >> (p. 795) Indicates whether or not an instance can move to the DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE (p. 715) state without being in the DDS_ALIVE_INSTANCE_STATE (p. 715) state.
DDS_DataReaderProtocolQosPolicy (p. 1596)	propagate_unregister_of_disposed_instances (p. 1599)
	<< extension >> (p. 795) Indicates whether or not an instance can move to the DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE (p. 715) state directly from the DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE (p. 715).

DDS_DataWriterProtocolQosPolicy (p. 1649)	virtual_guid (p. 1650)
	<< <i>extension</i> >> (p. 795) The virtual GUID (Global Unique Identifier).
DDS_DataWriterProtocolQosPolicy (p. 1649)	rtps_object_id (p. 1650)
	<< <i>extension</i> >> (p. 795) The RTPS Object ID.
DDS_DataWriterProtocolQosPolicy (p. 1649)	push_on_write (p. 1650)
	<< <i>extension</i> >> (p. 795) Whether to push sample out when write is called.
DDS_DataWriterProtocolQosPolicy (p. 1649)	disable_positive_acks (p. 1651)
	<< <i>extension</i> >> (p. 795) Controls whether or not the writer expects positive acknowledgements from matching readers.
DDS_DataWriterProtocolQosPolicy (p. 1649)	disable_inline_keyhash (p. 1651)
	<< <i>extension</i> >> (p. 795) Controls whether or not a keyhash is propagated on the wire with each sample.
DDS_DataWriterProtocolQosPolicy (p. 1649)	serialize_key_with_dispose (p. 1652)
	<< <i>extension</i> >> (p. 795) Controls whether or not the serialized key is propagated on the wire with dispose samples.
DDS_DataWriterProtocolQosPolicy (p. 1649)	propagate_app_ack_with_no_response (p. 1653)
	<< <i>extension</i> >> (p. 795) Controls whether or not a DDS_DataWriter (p. 485) receives DDS_DataWriterListener::on_application_acknowledgment (p. 1648) notifications with an empty or invalid response.
DDS_DataWriterProtocolQosPolicy (p. 1649)	rtps_reliable_writer (p. 1653)
	<< <i>extension</i> >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a DDS_DataWriter (p. 485). This parameter only has effect if both the writer and the matching reader are configured with DDS_RELIABLE_RELIABILITY_QOS (p. 1323) DDS_ReliabilityQosPolicyKind (p. 1322).
DDS_DataWriterProtocolQosPolicy (p. 1649)	initial_virtual_sequence_number (p. 1653)
	<< <i>extension</i> >> (p. 795) Determines, the initial virtual sequence number for this DataWriter.
DDS_SystemResourceLimitsQosPolicy (p. 2024)	max_objects_per_thread (p. 2025)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The maximum number of objects that can be stored per thread for a DDS_DomainParticipantFactory (p. 32).
DDS_SystemResourceLimitsQosPolicy (p. 2024)	initial_objects_per_thread (p. 2025)

	<< <i>extension</i> >> (p. 795) The number of objects per thread for a DDS_DomainParticipantFactory (p. 32) for which infrastructure will initially be allocated.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_writer_allocation (p. 1735)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to local DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_reader_allocation (p. 1735)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to local DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_publisher_allocation (p. 1736)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to local Publisher.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_subscriber_allocation (p. 1736)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to local Subscriber.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_topic_allocation (p. 1736)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to local Topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_writer_allocation (p. 1737)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to remote DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_reader_allocation (p. 1737)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to remote DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_participant_allocation (p. 1737)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to remote DomainParticipants.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	matching_writer_reader_pair_allocation (p. 1738)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to matching local writer and remote/local reader pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	matching_reader_writer_pair_allocation (p. 1738)

	<< extension >> (p. 795) Allocation settings applied to matching local reader and remote/local writer pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	ignored_entity_allocation (p. 1738)
	<< extension >> (p. 795) Allocation settings applied to ignored entities.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	content_filtered_topic_allocation (p. 1739)
	<< extension >> (p. 795) Allocation settings applied to content filtered topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	content_filter_allocation (p. 1739)
	<< extension >> (p. 795) Allocation settings applied to content filter.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	read_condition_allocation (p. 1739)
	<< extension >> (p. 795) Allocation settings applied to read condition pool.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	query_condition_allocation (p. 1740)
	<< extension >> (p. 795) Allocation settings applied to query condition pool.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	outstanding_asynchronous_sample_allocation (p. 1740)
	<< extension >> (p. 795) Allocation settings applied to the maximum number of samples (from all DDS_DataWriter (p. 485)) waiting to be asynchronously written.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	flow_controller_allocation (p. 1740)
	<< extension >> (p. 795) Allocation settings applied to flow controllers.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_writer_hash_buckets (p. 1741)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Hash_Buckets settings applied to local DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_reader_hash_buckets (p. 1741)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Number of hash buckets for local DataReaders.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_publisher_hash_buckets (p. 1741)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for local Publisher.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_subscriber_hash_buckets (p. 1742)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for local Subscriber.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_topic_hash_buckets (p. 1742)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for local Topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_writer_hash_buckets (p. 1742)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for remote DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_reader_hash_buckets (p. 1743)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for remote DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_participant_hash_buckets (p. 1743)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for remote DomainParticipants.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	matching_writer_reader_pair_hash_buckets (p. 1743)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for matching local writer and remote/local reader pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	matching_reader_writer_pair_hash_buckets (p. 1744)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for matching local reader and remote/local writer pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	ignored_entity_hash_buckets (p. 1744)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for ignored entities.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	content_filtered_topic_hash_buckets (p. 1744)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for content filtered topics.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	content_filter_hash_buckets (p. 1745)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for content filters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	flow_controller_hash_buckets (p. 1745)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for flow controllers.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	max_gather_destinations (p. 1745)
	<< <i>extension</i> >> (p. 795) Maximum number of destinations per RTI Connext send.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	participant_user_data_max_length (p. 1746)
	<< <i>extension</i> >> (p. 795) Maximum length of user data in DDS_DomainParticipantQos (p. 1726) and DDS_ParticipantBuiltinTopicData (p. 1870)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	topic_data_max_length (p. 1746)
	<< <i>extension</i> >> (p. 795) Maximum length of topic data in DDS_TopicQos (p. 2042), DDS_TopicBuiltinTopicData (p. 2034), DDS_PublicationBuiltinTopicData (p. 1905) and DDS_SubscriptionBuiltinTopicData (p. 2012)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	publisher_group_data_max_length (p. 1746)
	<< <i>extension</i> >> (p. 795) Maximum length of group data in DDS_PublisherQos (p. 1918) and DDS_PublicationBuiltinTopicData (p. 1905)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	subscriber_group_data_max_length (p. 1747)
	<< <i>extension</i> >> (p. 795) Maximum length of group data in DDS_SubscriberQos (p. 2010) and DDS_SubscriptionBuiltinTopicData (p. 2012)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	writer_user_data_max_length (p. 1747)
	<< <i>extension</i> >> (p. 795) Maximum length of user data in DDS_DataWriterQos (p. 1666) and DDS_PublicationBuiltinTopicData (p. 1905)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	reader_user_data_max_length (p. 1747)

	<< <i>extension</i> >> (p. 795) Maximum length of user data in DDS_DataReaderQos (p. 1611) and DDS_↔SubscriptionBuiltinTopicData (p. 2012)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	max_partitions (p. 1748)
	<< <i>extension</i> >> (p. 795) Maximum number of partition name strings allowable in a DDS_PartitionQosPolicy (p. 1881).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	max_partition_cumulative_characters (p. 1748)
	<< <i>extension</i> >> (p. 795) Maximum number of combined characters allowable in all partition names in a DDS_PartitionQosPolicy (p. 1881).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	type_code_max_serialized_length (p. 1749)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Maximum size of serialized string for type code.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	type_object_max_serialized_length (p. 1749)
	<< <i>extension</i> >> (p. 795) The maximum length, in bytes, that the buffer to serialize a TypeObject can consume.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	type_object_max_deserialized_length (p. 1750)
	<< <i>extension</i> >> (p. 795) The maximum number of bytes that a deserialized TypeObject can consume.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	deserialized_type_object_dynamic_allocation_↔threshold (p. 1751)
	<< <i>extension</i> >> (p. 795) A threshold, in bytes, for dynamic memory allocation for the deserialized Type↔Object.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	serialized_type_object_dynamic_allocation_↔threshold (p. 1750)
	<< <i>extension</i> >> (p. 795) A threshold, in bytes, for dynamic memory allocation for the serialized TypeObject.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	contentfilter_property_max_length (p. 1751)
	<< <i>extension</i> >> (p. 795) This field is the maximum length of all data related to a Content-filtered topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	channel_seq_max_length (p. 1752)

	<< <i>extension</i> >> (p. 795) Maximum number of channels that can be specified in DDS_MultiChannelQosPolicy (p. 1856) for MultiChannel DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	channel_filter_expression_max_length (p. 1752)
	<< <i>extension</i> >> (p. 795) Maximum length of a channel DDS_ChannelSettings_t::filter_expression (p. 1564) in a MultiChannel DataWriter.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	participant_property_list_max_length (p. 1752)
	<< <i>extension</i> >> (p. 795) Maximum number of properties associated with the DDS_DomainParticipant (p. 77).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	participant_property_string_max_length (p. 1753)
	<< <i>extension</i> >> (p. 795) Maximum string length of the properties associated with the DDS_DomainParticipant (p. 77).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	writer_property_list_max_length (p. 1753)
	<< <i>extension</i> >> (p. 795) Maximum number of properties associated with a DDS_DataWriter (p. 485).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	writer_property_string_max_length (p. 1753)
	<< <i>extension</i> >> (p. 795) Maximum string length of the properties associated with a DDS_DataWriter (p. 485).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	reader_property_list_max_length (p. 1754)
	<< <i>extension</i> >> (p. 795) Maximum number of properties associated with a DDS_DataReader (p. 615).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	reader_property_string_max_length (p. 1754)
	<< <i>extension</i> >> (p. 795) Maximum string length of the properties associated with a DDS_DataReader (p. 615).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	max_endpoint_groups (p. 1754)
	<< <i>extension</i> >> (p. 795) Maximum number of DDS_EndpointGroup_t (p. 1784) allowable in a DDS_AvailabilityQosPolicy (p. 1547).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	max_endpoint_group_cumulative_characters (p. 1755)

	<< <i>extension</i> >> (p. 795) Maximum number of combined <code>role_name</code> characters allowed in all DDS_EndpointGroup_t (p. 1784) in a DDS_AvailabilityQosPolicy (p. 1547).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	transport_info_list_max_length (p. 1755)
	<< <i>extension</i> >> (p. 795) Maximum number of installed transports to send and receive information about in DDS_ParticipantBuiltinTopicData::transport_info (p. 1873).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	ignored_entity_replacement_kind (p. 1755)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Replacement policy for the ignored entities. It sets what entity can be replaced when resource limits set in DDS_DomainParticipantResourceLimitsQosPolicy::ignored_entity_allocation (p. 1738) are reached.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_topic_query_allocation (p. 1756)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to remote TopicQueries.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_topic_query_hash_buckets (p. 1756)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for remote TopicQueries.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	writer_data_tag_list_max_length (p. 1757)
	<< <i>extension</i> >> (p. 795) Maximum number of data tags associated with a DDS_DataWriter (p. 485).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	writer_data_tag_string_max_length (p. 1757)
	<< <i>extension</i> >> (p. 795) Maximum string length of the data tags associated with a DDS_DataWriter (p. 485).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	reader_data_tag_list_max_length (p. 1757)
	<< <i>extension</i> >> (p. 795) Maximum number of data tags associated with a DDS_DataReader (p. 615).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	reader_data_tag_string_max_length (p. 1758)
	<< <i>extension</i> >> (p. 795) Maximum string length of the data tags associated with a DDS_DataReader (p. 615).

DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	shmem_ref_transfer_mode_max_segments (p. 1758)
	<< <i>extension</i> >> (p. 795) Maximum number of segments created by all DataWriters belonging to a DDS_DomainParticipant (p. 77).
DDS_EventQosPolicy (p. 1793)	thread (p. 1793)
	<< <i>extension</i> >> (p. 795) Event thread QoS.
DDS_EventQosPolicy (p. 1793)	initial_count (p. 1794)
	<< <i>extension</i> >> (p. 795) The initial number of events.
DDS_EventQosPolicy (p. 1793)	max_count (p. 1794)
	<< <i>extension</i> >> (p. 795) The maximum number of events.
DDS_DatabaseQosPolicy (p. 1581)	thread (p. 1582)
	<< <i>extension</i> >> (p. 795) Database thread settings.
DDS_DatabaseQosPolicy (p. 1581)	shutdown_timeout (p. 1583)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The maximum wait time during a shutdown.
DDS_DatabaseQosPolicy (p. 1581)	cleanup_period (p. 1583)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The period at which the database thread wakes up to remove deleted records.
DDS_DatabaseQosPolicy (p. 1581)	shutdown_cleanup_period (p. 1583)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The clean-up period used during database shut-down.
DDS_DatabaseQosPolicy (p. 1581)	initial_records (p. 1584)
	<< <i>extension</i> >> (p. 795) The initial number of total records.
DDS_DatabaseQosPolicy (p. 1581)	max_skiplist_level (p. 1584)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The maximum level of the skiplist.
DDS_DatabaseQosPolicy (p. 1581)	initial_weak_references (p. 1585)
	<< <i>extension</i> >> (p. 795) The initial number of weak references.
DDS_DatabaseQosPolicy (p. 1581)	max_weak_references (p. 1584)
	<< <i>extension</i> >> (p. 795) The maximum number of weak references.
DDS_ReceiverPoolQosPolicy (p. 1934)	thread (p. 1935)
	<< <i>extension</i> >> (p. 795) Receiver pool thread(s).

DDS_ReceiverPoolQosPolicy (p. 1934)	buffer_size (p. 1935)
	<< <i>extension</i> >> (p. 795) The receive buffer size in bytes.
DDS_ReceiverPoolQosPolicy (p. 1934)	buffer_alignment (p. 1936)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The receive buffer alignment.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_liveliness_lease_duration (p. 1692)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) The liveliness lease duration for the participant.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_liveliness_assert_period (p. 1692)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) The period to assert liveliness for the participant.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_announcement_period (p. 1693)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) The period at which a participant announces itself to potential peers when using the Simple Participant Discovery Protocol 2.0 (SPDP2).
DDS_DiscoveryConfigQosPolicy (p. 1689)	remote_participant_purge_kind (p. 1693)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.
DDS_DiscoveryConfigQosPolicy (p. 1689)	max_liveliness_loss_detection_period (p. 1694)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) 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. 1689)	initial_participant_announcements (p. 1694)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) The number of initial announcements sent when a participant is first enabled.
DDS_DiscoveryConfigQosPolicy (p. 1689)	new_remote_participant_announcements (p. 1695)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) The number of participant announcements sent when a remote participant is newly discovered.
DDS_DiscoveryConfigQosPolicy (p. 1689)	min_initial_participant_announcement_period (p. 1695)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) The minimum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.

DDS_DiscoveryConfigQosPolicy (p. 1689)	max_initial_participant_announcement_period (p. 1695)
	<< basic >> (p. 783) << extension >> (p. 795) The maximum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_reader_resource_limits (p. 1696)
	<< extension >> (p. 795) Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 1689)	publication_reader (p. 1696)
	<< extension >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in publication reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	publication_reader_resource_limits (p. 1697)
	<< extension >> (p. 795) Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 1689)	subscription_reader (p. 1697)
	<< extension >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in subscription reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	subscription_reader_resource_limits (p. 1698)
	<< extension >> (p. 795) Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_message_writer (p. 1702)
	<< extension >> (p. 795) 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. 1323) DDS_ReliabilityQos↵PolicyKind (p. 1322).
DDS_DiscoveryConfigQosPolicy (p. 1689)	publication_writer_publish_mode (p. 1703)
	<< extension >> (p. 795) Publish mode policy for the built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	subscription_writer_publish_mode (p. 1704)
	<< extension >> (p. 795) Publish mode policy for the built-in subscription writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	asynchronous_publisher (p. 1704)
	<< extension >> (p. 795) Asynchronous publishing settings for the discovery DDS_Publisher (p. 443) and all entities that are created by it.
DDS_DiscoveryConfigQosPolicy (p. 1689)	default_domain_announcement_period (p. 1704)

	<< <i>extension</i> >> (p. 795) The period to announce a participant to the default domain 0.
DDS_DiscoveryConfigQosPolicy (p. 1689)	ignore_default_domain_announcements (p. 1705)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) 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. 1689)	participant_message_reader_reliability_kind (p. 1701)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Reliability policy for a built-in participant message reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_message_reader (p. 1702)
	<< <i>extension</i> >> (p. 795) 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. 1701) is set to DDS_RELIABLE_RELIABILITY_QOS (p. 1323).
DDS_DiscoveryConfigQosPolicy (p. 1689)	publication_writer (p. 1698)
	<< <i>extension</i> >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	publication_writer_data_lifecycle (p. 1699)
	<< <i>extension</i> >> (p. 795) Writer data lifecycle settings for a built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	subscription_writer (p. 1700)
	<< <i>extension</i> >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in subscription writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	subscription_writer_data_lifecycle (p. 1700)
	<< <i>extension</i> >> (p. 795) Writer data lifecycle settings for a built-in subscription writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	builtin_discovery_plugins (p. 1701)
	<< <i>extension</i> >> (p. 795) Mask of built-in discovery plugin kinds.
DDS_DiscoveryConfigQosPolicy (p. 1689)	enabled_builtin_channels (p. 1701)
	<< <i>extension</i> >> (p. 795) The mask specifying which built-in channels should be enabled.
DDS_DiscoveryConfigQosPolicy (p. 1689)	service_request_writer (p. 1705)

	<< <i>extension</i> >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in DDS_ServiceRequest (p. 2001) writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	service_request_writer_data_lifecycle (p. 1706)
	<< <i>extension</i> >> (p. 795) Writer data lifecycle settings for a built-in DDS_ServiceRequest (p. 2001) writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	service_request_writer_publish_mode (p. 1706)
	<< <i>extension</i> >> (p. 795) Publish mode policy for the built-in service request writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	service_request_reader (p. 1707)
	<< <i>extension</i> >> (p. 795) RTPS reliable reader protocol-related configuration settings for a built-in DDS_ServiceRequest (p. 2001) reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	locator_reachability_lease_duration (p. 1708)
	<< <i>extension</i> >> (p. 795) 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. 1689)	locator_reachability_assert_period (p. 1707)
	<< <i>extension</i> >> (p. 795) Period at which this DomainParticipant will assert the locators discovered from other DomainParticipants.
DDS_DiscoveryConfigQosPolicy (p. 1689)	locator_reachability_change_detection_period (p. 1708)
	<< <i>extension</i> >> (p. 795) Period at which this DomainParticipant will check if its locators are reachable from other DomainParticipants.
DDS_DiscoveryConfigQosPolicy (p. 1689)	secure_volatile_writer (p. 1709)
	<< <i>extension</i> >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in secure volatile writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	secure_volatile_writer_publish_mode (p. 1710)
	<< <i>extension</i> >> (p. 795) Publish mode policy for the built-in secure volatile writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	secure_volatile_reader (p. 1710)
	<< <i>extension</i> >> (p. 795) RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	endpoint_type_object_lb_serialization_threshold (p. 1711)

	<< extension >> (p. 795) Option to reduce the size required to propagate a TypeObject in Simple Endpoint Discovery
DDS_DiscoveryConfigQosPolicy (p. 1689)	dns_tracker_polling_period (p. 1711)
	<< extension >> (p. 795) 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. 1689)	participant_configuration_reader (p. 1714)
	<< extension >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in participant configuration reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_configuration_reader_resource_limits (p. 1714)
	<< extension >> (p. 795) Resource limits for the built-in topic participant configuration reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_configuration_writer_publish_mode (p. 1712)
	<< extension >> (p. 795) Publish mode policy for the built-in participant configuration writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_configuration_writer (p. 1712)
	<< extension >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in participant configuration writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_configuration_writer_data_lifecycle (p. 1713)
	<< extension >> (p. 795) Writer data lifecycle settings for a built-in participant configuration writer.
DDS_RtpsWellKnownPorts_t (p. 1978)	port_base (p. 1980)
	<< extension >> (p. 795) The base port offset.
DDS_RtpsWellKnownPorts_t (p. 1978)	domain_id_gain (p. 1981)
	<< extension >> (p. 795) Tunable domain gain parameter.
DDS_RtpsWellKnownPorts_t (p. 1978)	participant_id_gain (p. 1982)
	<< extension >> (p. 795) Tunable participant gain parameter.
DDS_RtpsWellKnownPorts_t (p. 1978)	builtin_multicast_port_offset (p. 1982)
	<< extension >> (p. 795) Additional offset for meta-traffic multicast port.
DDS_RtpsWellKnownPorts_t (p. 1978)	builtin_unicast_port_offset (p. 1982)

	<< <i>extension</i> >> (p. 795) Additional offset for meta-traffic unicast port.
DDS_RtpsWellKnownPorts_t (p. 1978)	user_multicast_port_offset (p. 1983)
	<< <i>extension</i> >> (p. 795) Additional offset for usertraf- fic multicast port.
DDS_RtpsWellKnownPorts_t (p. 1978)	user_unicast_port_offset (p. 1983)
	<< <i>extension</i> >> (p. 795) Additional offset for usertraf- fic unicast port.
DDS_RtpsReliableReaderProtocol_t (p. 1954)	min_heartbeat_response_delay (p. 1955)
	<< <i>extension</i> >> (p. 795) The minimum delay to re- spond to a heartbeat.
DDS_RtpsReliableReaderProtocol_t (p. 1954)	max_heartbeat_response_delay (p. 1955)
	<< <i>extension</i> >> (p. 795) The maximum delay to re- spond to a heartbeat.
DDS_RtpsReliableReaderProtocol_t (p. 1954)	heartbeat_suppression_duration (p. 1956)
	<< <i>extension</i> >> (p. 795) The duration a reader ig- nores consecutively received heartbeats
DDS_RtpsReliableReaderProtocol_t (p. 1954)	nack_period (p. 1956)
	<< <i>extension</i> >> (p. 795) The period at which to send NACKs
DDS_RtpsReliableReaderProtocol_t (p. 1954)	receive_window_size (p. 1957)
	<< <i>extension</i> >> (p. 795) The number of received out- of-order samples a reader can keep at a time
DDS_RtpsReliableReaderProtocol_t (p. 1954)	round_trip_time (p. 1957)
	<< <i>extension</i> >> (p. 795) The duration from sending a NACK to receiving a repair of a sample
DDS_RtpsReliableReaderProtocol_t (p. 1954)	app_ack_period (p. 1958)
	<< <i>extension</i> >> (p. 795) The period at which application-level acknowledgment messages are sent
DDS_RtpsReliableReaderProtocol_t (p. 1954)	min_app_ack_response_keep_duration (p. 1958)
	<< <i>extension</i> >> (p. 795) Minimum duration for which application-level acknowledgment response data is kept
DDS_RtpsReliableReaderProtocol_t (p. 1954)	samples_per_app_ack (p. 1959)
	<< <i>extension</i> >> (p. 795) The minimum number of samples acknowledged by one application-level ac- knowledgment message
DDS_RtpsReliableWriterProtocol_t (p. 1960)	low_watermark (p. 1961)

	<< <i>extension</i> >> (p. 795) 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. 1234) is considered to have changed.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	high_watermark (p. 1962)
	<< <i>extension</i> >> (p. 795) 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. 1234) is considered to have changed.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	heartbeat_period (p. 1963)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) The period at which to send heartbeats.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	fast_heartbeat_period (p. 1963)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) An alternative heartbeat period used when a reliable writer needs to flush its unacknowledged samples more quickly.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	late_joiner_heartbeat_period (p. 1964)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) 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. 1960)	virtual_heartbeat_period (p. 1965)
	<< <i>extension</i> >> (p. 795) 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. 1960)	samples_per_virtual_heartbeat (p. 1965)
	<< <i>extension</i> >> (p. 795) The number of samples that a reliable writer has to publish before sending a virtual heartbeat.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	max_heartbeat_retries (p. 1966)
	<< <i>extension</i> >> (p. 795) The maximum number of <i>periodic</i> heartbeat retries before marking a remote reader as inactive.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	inactivate_nonprogressing_readers (p. 1966)
	<< <i>extension</i> >> (p. 795) Whether to treat remote readers as inactive when their NACKs do not progress
DDS_RtpsReliableWriterProtocol_t (p. 1960)	heartbeats_per_max_samples (p. 1967)

	<< <i>extension</i> >> (p. 795) The number of piggyback heartbeats sent per max send window.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	min_nack_response_delay (p. 1968)
	<< <i>extension</i> >> (p. 795) The minimum delay to respond to a NACK or NACK_FRAG.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	max_nack_response_delay (p. 1968)
	<< <i>extension</i> >> (p. 795) The maximum delay to respond to a NACK or NACK_FRAG.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	nack_suppression_duration (p. 1969)
	<< <i>extension</i> >> (p. 795) The duration for ignoring consecutive NACKs/NACK_FRAGs that may trigger redundant repairs.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	max_bytes_per_nack_response (p. 1969)
	<< <i>extension</i> >> (p. 795) The maximum total message size when resending rejected samples or data fragments.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_enable_adaptive_sample_↔keep_duration (p. 1971)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Enables dynamic adjustment of sample keep duration in response to congestion.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_min_sample_keep_duration (p. 1970)
	<< <i>extension</i> >> (p. 795) The minimum duration a sample is queued for ACK-disabled readers.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_max_sample_keep_duration (p. 1971)
	<< <i>extension</i> >> (p. 795) The maximum duration a sample is queued for ACK-disabled readers.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_decrease_sample_keep_↔duration_factor (p. 1972)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Controls rate of contraction of dynamic sample keep duration.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_increase_sample_keep_↔duration_factor (p. 1972)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Controls rate of growth of dynamic sample keep duration.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	min_send_window_size (p. 1973)

	<< basic >> (p. 783) << extension >> (p. 795) Minimum size of send window of unacknowledged samples
DDS_RtpsReliableWriterProtocol_t (p. 1960)	max_send_window_size (p. 1973)
	<< basic >> (p. 783) << extension >> (p. 795) Maximum size of send window of unacknowledged samples
DDS_RtpsReliableWriterProtocol_t (p. 1960)	send_window_update_period (p. 1974)
	<< extension >> (p. 795) Period in which send window may be dynamically changed
DDS_RtpsReliableWriterProtocol_t (p. 1960)	send_window_increase_factor (p. 1975)
	<< extension >> (p. 795) Increases send window size by this percentage when reacting dynamically to network conditions
DDS_RtpsReliableWriterProtocol_t (p. 1960)	send_window_decrease_factor (p. 1976)
	<< extension >> (p. 795) Decreases send window size by this percentage when reacting dynamically to network conditions
DDS_RtpsReliableWriterProtocol_t (p. 1960)	enable_multicast_periodic_heartbeat (p. 1977)
	<< extension >> (p. 795) Whether periodic heartbeat messages are sent over multicast
DDS_RtpsReliableWriterProtocol_t (p. 1960)	multicast_resend_threshold (p. 1977)
	<< extension >> (p. 795) The minimum number of requesting readers needed to trigger a multicast resend
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_repair_piggyback_heartbeat (p. 1978)
	<< extension >> (p. 795) Prevents piggyback heartbeats from being sent with repair samples
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	initial_samples (p. 1558)
	<< extension >> (p. 795) Initial number of samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_samples (p. 1558)
	<< extension >> (p. 795) Maximum number of samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	initial_infos (p. 1559)
	<< extension >> (p. 795) Initial number of sample infos.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_infos (p. 1559)
	<< extension >> (p. 795) Maximum number of sample infos.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	initial_outstanding_reads (p. 1559)

	<< <i>extension</i> >> (p. 795) The initial number of outstanding reads that have not called finish yet on the same built-in topic DDS_DataReader (p. 615).
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_outstanding_reads (p. 1560)
	<< <i>extension</i> >> (p. 795) The maximum number of outstanding reads that have not called finish yet on the same built-in topic DDS_DataReader (p. 615).
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_samples_per_read (p. 1560)
	<< <i>extension</i> >> (p. 795) Maximum number of samples that can be read/taken on a same built-in topic DDS_DataReader (p. 615).
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	disable_fragmentation_support (p. 1560)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Determines whether the built-in topic DDS_DataReader (p. 615) can receive fragmented samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_fragmented_samples (p. 1561)
	<< <i>extension</i> >> (p. 795) The maximum number of samples for which the built-in topic DDS_DataReader (p. 615) may store fragments at a given point in time.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	initial_fragmented_samples (p. 1561)
	<< <i>extension</i> >> (p. 795) The initial number of samples for which a built-in topic DDS_DataReader (p. 615) may store fragments.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_fragmented_samples_per_remote_writer (p. 1562)
	<< <i>extension</i> >> (p. 795) The maximum number of samples per remote writer for which a built-in topic DDS_DataReader (p. 615) may store fragments.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_fragments_per_sample (p. 1562)
	<< <i>extension</i> >> (p. 795) Maximum number of fragments for a single sample.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	dynamically_allocate_fragmented_samples (p. 1563)
	<< <i>extension</i> >> (p. 795) Determines whether the built-in topic DDS_DataReader (p. 615) pre-allocates storage for storing fragmented samples.
DDS_TransportUnicastSettings_t (p. 2067)	transports (p. 2068)
	<< <i>extension</i> >> (p. 795) A sequence of transport aliases that specifies the unicast interfaces on which to receive <i>unicast</i> traffic for the entity.
DDS_TransportUnicastSettings_t (p. 2067)	receive_port (p. 2068)

	<< extension >> (p. 795) The unicast port on which the entity can receive data.
DDS_TransportMulticastSettings_t (p. 2060)	transports (p. 2061)
	<< extension >> (p. 795) A sequence of transport aliases that specifies the transports on which to receive <i>multicast</i> traffic for the entity.
DDS_TransportMulticastSettings_t (p. 2060)	receive_address (p. 2061)
	<< extension >> (p. 795) The multicast group address on which the entity can receive data.
DDS_TransportMulticastSettings_t (p. 2060)	receive_port (p. 2061)
	<< extension >> (p. 795) The multicast port on which the entity can receive data.
DDS_TransportMulticastMapping_t (p. 2053)	addresses (p. 2054)
	<< extension >> (p. 795) A string containing a comma-separated list of IP addresses or IP address ranges to be used to receive <i>multicast</i> traffic for the entity with a topic that matches the DDS_TransportMulticastMapping_t::topic_expression (p. 2054).
DDS_TransportMulticastMapping_t (p. 2053)	topic_expression (p. 2054)
	<< extension >> (p. 795) A regular expression that will be used to map topic names to corresponding multicast receive addresses.
DDS_TransportMulticastMapping_t (p. 2053)	mapping_function (p. 2055)
	<< extension >> (p. 795) 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. 1901)	value (p. 1903)
	<< basic >> (p. 783) << extension >> (p. 795) Sequence of properties.
DDS_EntityNameQosPolicy (p. 1790)	name (p. 1790)
	<< basic >> (p. 783) << extension >> (p. 795) The name of the entity.
DDS_EntityNameQosPolicy (p. 1790)	role_name (p. 1791)
	<< extension >> (p. 795) The entity role name.
DDS_ServiceQosPolicy (p. 2000)	kind (p. 2000)
	<< extension >> (p. 795) The kind of service.
DDS_BatchQosPolicy (p. 1552)	enable (p. 1552)
	<< basic >> (p. 783) << extension >> (p. 795) Specifies whether or not batching is enabled.

DDS_BatchQosPolicy (p. 1552)	max_data_bytes (p. 1553)
	<< basic >> (p. 783) << extension >> (p. 795) The maximum cumulative length of all serialized samples in a batch.
DDS_BatchQosPolicy (p. 1552)	max_samples (p. 1553)
	<< basic >> (p. 783) << extension >> (p. 795) The maximum number of samples in a batch.
DDS_BatchQosPolicy (p. 1552)	max_flush_delay (p. 1554)
	<< extension >> (p. 795) The maximum flush delay.
DDS_BatchQosPolicy (p. 1552)	source_timestamp_resolution (p. 1554)
	<< extension >> (p. 795) Batch source timestamp resolution.
DDS_BatchQosPolicy (p. 1552)	thread_safe_write (p. 1555)
	<< extension >> (p. 795) Determines whether or not the write operation is thread safe.
DDS_ProfileQosPolicy (p. 1898)	string_profile (p. 1899)
	<< basic >> (p. 783) << extension >> (p. 795) Sequence of strings containing a XML document to load.
DDS_ProfileQosPolicy (p. 1898)	url_profile (p. 1899)
	<< basic >> (p. 783) << extension >> (p. 795) Sequence of URL groups (p. 973) containing a set of XML documents to load.
DDS_ProfileQosPolicy (p. 1898)	ignore_user_profile (p. 1899)
	<< basic >> (p. 783) << extension >> (p. 795) Ignores the file USER_QOS_PROFILES.xml in the current working directory.
DDS_ProfileQosPolicy (p. 1898)	ignore_environment_profile (p. 1899)
	<< basic >> (p. 783) << extension >> (p. 795) Ignores the value of the NDDS_QOS_PROFILES (p. 973).
DDS_ProfileQosPolicy (p. 1898)	ignore_resource_profile (p. 1900)
	<< basic >> (p. 783) << extension >> (p. 795) Ignores the file NDDS_QOS_PROFILES.xml.
DDS_LoggingQosPolicy (p. 1831)	verbosity (p. 1832)
	<< basic >> (p. 783) << extension >> (p. 795) The verbosity at which RTI Connex diagnostic information is logged.
DDS_LoggingQosPolicy (p. 1831)	category (p. 1832)

	<< basic >> (p. 783) << extension >> (p. 795) Categories of logged messages.
DDS_LoggingQosPolicy (p. 1831)	print_format (p. 1832)
	<< extension >> (p. 795) The format used to output RTI Connex diagnostic information.
DDS_LoggingQosPolicy (p. 1831)	output_file (p. 1832)
	<< basic >> (p. 783) << extension >> (p. 795) Specifies the file to which log messages will be redirected to.
DDS_LoggingQosPolicy (p. 1831)	output_file_suffix (p. 1833)
	<< basic >> (p. 783) << extension >> (p. 795) Sets the file suffix when logging to a set of files
DDS_LoggingQosPolicy (p. 1831)	max_bytes_per_file (p. 1833)
	<< extension >> (p. 795) Specifies the maximum number of bytes a single file can contain
DDS_LoggingQosPolicy (p. 1831)	max_files (p. 1834)
	<< extension >> (p. 795) Specifies the maximum number of files to create before overwriting the previous ones.
DDS_MonitoringEventDistributionSettings (p. 1840)	concurrency_level (p. 1841)
	<< extension >> (p. 795) Defines how concurrent the push of event metrics to RTI Monitoring Library 2.0 is.
DDS_MonitoringEventDistributionSettings (p. 1840)	datawriter_qos_profile_name (p. 1842)
	<< extension >> (p. 795) The fully qualified name of the profile used to configure the DDS_DataWriter (p. 485) that distributes event metrics.
DDS_MonitoringEventDistributionSettings (p. 1840)	thread (p. 1842)
	<< extension >> (p. 795) The settings of the event metric thread.
DDS_MonitoringEventDistributionSettings (p. 1840)	publication_period (p. 1843)
	<< basic >> (p. 783) << extension >> (p. 795) Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.
DDS_MonitoringPeriodicDistributionSettings (p. 1852)	datawriter_qos_profile_name (p. 1852)
	<< extension >> (p. 795) The fully qualified name of the profile used to configure the DDS_DataWriter (p. 485) that distributes periodic metrics.
DDS_MonitoringPeriodicDistributionSettings (p. 1852)	thread (p. 1852)

	<< extension >> (p. 795) The settings of the periodic metric thread.
DDS_MonitoringPeriodicDistributionSettings (p. 1852)	polling_period (p. 1853)
	<< basic >> (p. 783) << extension >> (p. 795) Period at which the periodic metric thread polls and publishes the periodic metrics.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	concurrency_level (p. 1844)
	<< extension >> (p. 795) Defines how concurrent the push of log messages to RTI Monitoring Library 2.0 is.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	max_historical_logs (p. 1845)
	<< basic >> (p. 783) << extension >> (p. 795) The number of log messages that RTI Monitoring Library 2.0 will keep as history.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	datawriter_qos_profile_name (p. 1845)
	<< extension >> (p. 795) The fully qualified name of the profile used to configure the DDS_DataWriter (p. 485) that distributes log messages.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	thread (p. 1846)
	<< extension >> (p. 795) The settings of the logging thread.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	publication_period (p. 1846)
	<< basic >> (p. 783) << extension >> (p. 795) Period at which the logging thread publishes log messages.
DDS_MonitoringDedicatedParticipantSettings (p. 1836)	enable (p. 1836)
	<< basic >> (p. 783) << extension >> (p. 795) Enables the use of a dedicated DDS_DomainParticipant (p. 77) to distribute the RTI Connex application telemetry data.
DDS_MonitoringDedicatedParticipantSettings (p. 1836)	domain_id (p. 1837)
	<< basic >> (p. 783) << extension >> (p. 795) The domain ID used in the creation of RTI Monitoring Library 2.0 DDS_DomainParticipant (p. 77).

DDS_MonitoringDedicatedParticipantSettings (p. 1836)	participant_qos_profile_name (p. 1837)
	<< <i>extension</i> >> (p. 795) The fully qualified name of the profile used to configure the DDS_DomainParticipant (p. 77) that will be used to distribute telemetry data.
DDS_MonitoringDedicatedParticipantSettings (p. 1836)	collector_initial_peers (p. 1837)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) 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. 1839).
DDS_MonitoringDistributionSettings (p. 1838)	dedicated_participant (p. 1839)
	<< <i>extension</i> >> (p. 795) Configures the use of a dedicated DDS_DomainParticipant (p. 77) to distribute the RTI Connext application telemetry data.
DDS_MonitoringDistributionSettings (p. 1838)	publisher_qos_profile_name (p. 1839)
	<< <i>extension</i> >> (p. 795) The fully qualified name of the profile used to configure the Publishers that distribute telemetry data.
DDS_MonitoringDistributionSettings (p. 1838)	event_settings (p. 1839)
	<< <i>extension</i> >> (p. 795) Configures the distribution of event metrics.
DDS_MonitoringDistributionSettings (p. 1838)	periodic_settings (p. 1840)
	<< <i>extension</i> >> (p. 795) Configures the distribution of periodic metrics.
DDS_MonitoringDistributionSettings (p. 1838)	logging_settings (p. 1840)
	<< <i>extension</i> >> (p. 795) Configures the distribution of logging messages.
DDS_MonitoringMetricSelection (p. 1849)	resource_selection (p. 1849)
	<< <i>extension</i> >> (p. 795) An expression pattern that selects a subset of resources by matching the pattern to the resource names. DDS_MonitoringMetricSelection::enabled_metrics_selection (p. 1850) and DDS_MonitoringMetricSelection::disabled_metrics_selection (p. 1850) are applied to this subset of resources.
DDS_MonitoringMetricSelection (p. 1849)	enabled_metrics_selection (p. 1850)

	<< extension >> (p. 795) A sequence of POSIX fn-match 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. 1849).
DDS_MonitoringMetricSelection (p. 1849)	disabled_metrics_selection (p. 1850)
	<< extension >> (p. 795) A sequence of POSIX fn-match 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. 1849).
DDS_MonitoringLoggingForwardingSettings (p. 1847)	middleware_forwarding_level (p. 1847)
	<< basic >> (p. 783) << extension >> (p. 795) Log messages with NDDS_CONFIG_LOG_FACILITY_MIDDLEWARE (p. 1447) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1847)	security_event_forwarding_level (p. 1847)
	<< basic >> (p. 783) << extension >> (p. 795) Log messages with NDDS_CONFIG_LOG_FACILITY_SECURITY_EVENT (p. 1447) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1847)	service_forwarding_level (p. 1848)
	<< basic >> (p. 783) << extension >> (p. 795) Log messages with NDDS_CONFIG_LOG_FACILITY_SERVICE (p. 1447) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1847)	user_forwarding_level (p. 1848)
	<< basic >> (p. 783) << extension >> (p. 795) Log messages with NDDS_CONFIG_LOG_FACILITY_USER (p. 1447) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringTelemetryData (p. 1855)	metrics (p. 1855)
	<< basic >> (p. 783) << extension >> (p. 795) Sequence of DDS_MonitoringMetricSelection (p. 1849) containing the event and periodic metrics that will be collected and distributed for a given set of observable resources.

DDS_MonitoringTelemetryData (p. 1855)	logs (p. 1856)
	<< basic >> (p. 783) << extension >> (p. 795) DDS_MonitoringLoggingForwardingSettings (p. 1847) containing the NDDS_Config_Syslog_Verbosity (p. 1447) levels that will be forwarded for the different NDDS_Config_LogFacility (p. 1446).
DDS_MonitoringQosPolicy (p. 1853)	enable (p. 1854)
	<< basic >> (p. 783) << extension >> (p. 795) Enables the collection and distribution of telemetry data for an RTI Connext application using RTI Monitoring Library 2.0.
DDS_MonitoringQosPolicy (p. 1853)	application_name (p. 1854)
	<< basic >> (p. 783) << extension >> (p. 795) The name of the resource that represents this RTI Connext application.
DDS_MonitoringQosPolicy (p. 1853)	distribution_settings (p. 1854)
	<< extension >> (p. 795) Configures the distribution of telemetry data.
DDS_MonitoringQosPolicy (p. 1853)	telemetry_data (p. 1855)
	<< extension >> (p. 795) Configures the telemetry data that will be distributed.
DDS_LocatorFilter_t (p. 1828)	locators (p. 1828)
	<< extension >> (p. 795) Sequence containing from one to 16 DDS_Locator_t (p. 1826), used to specify the multicast address locators of an individual channel within a MultiChannel DataWriter.
DDS_LocatorFilter_t (p. 1828)	filter_expression (p. 1828)
	<< extension >> (p. 795) A logical expression used to determine the data that will be published in the channel.
DDS_LocatorFilterQosPolicy (p. 1829)	locator_filters (p. 1829)
	<< extension >> (p. 795) A sequence of DDS_LocatorFilter_t (p. 1828). Each DDS_LocatorFilter_t (p. 1828) reports the configuration of a single channel of a MultiChannel DataWriter.
DDS_LocatorFilterQosPolicy (p. 1829)	filter_name (p. 1829)
	<< extension >> (p. 795) Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.
DDS_ChannelSettings_t (p. 1564)	multicast_settings (p. 1564)
	<< extension >> (p. 795) A sequence of DDS_TransportMulticastSettings_t (p. 2060) used to configure the multicast addresses associated with a channel.

DDS_ChannelSettings_t (p. 1564)	filter_expression (p. 1564)
	<< extension >> (p. 795) A logical expression used to determine the data that will be published in the channel.
DDS_ChannelSettings_t (p. 1564)	priority (p. 1565)
	<< extension >> (p. 795) Publication priority
DDS_MultiChannelQosPolicy (p. 1856)	channels (p. 1858)
	<< extension >> (p. 795) A sequence of DDS_ChannelSettings_t (p. 1564) used to configure the channels' properties. If the length of the sequence is zero, the QoS policy will be ignored.
DDS_MultiChannelQosPolicy (p. 1856)	filter_name (p. 1858)
	<< extension >> (p. 795) Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.
DDS_TopicQueryDispatchQosPolicy (p. 2047)	enable (p. 2048)
	<< extension >> (p. 795) Allows this writer to dispatch TopicQueries
DDS_TopicQueryDispatchQosPolicy (p. 2047)	publication_period (p. 2048)
	<< basic >> (p. 783) << extension >> (p. 795) Sets the periodic interval at which samples are published.
DDS_TopicQueryDispatchQosPolicy (p. 2047)	samples_per_period (p. 2049)
	<< extension >> (p. 795) Sets the maximum number of samples to publish in each publication_period.

5.49 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. 871), 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 Connex.

- **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 Connex 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 Connex 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 Connex 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 Connex.

- **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.

5.49.1 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. 871), then looking at any QoS in that category will be helpful.

5.50 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. 1936)	acknowledgment_kind (p. 1939)
	<< <i>extension</i> >> (p. 795) Kind of reliable acknowledgment
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_app_ack_response_length (p. 1633)
	<< <i>extension</i> >> (p. 795) Maximum length of application-level acknowledgment response data
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_app_ack_remote_readers (p. 1680)
	<< <i>extension</i> >> (p. 795) The maximum number of application-level acknowledging remote readers supported by a DDS_DataWriter (p. 485)
DDS_DataWriterProtocolQosPolicy (p. 1649)	propagate_app_ack_with_no_response (p. 1653)
	<< <i>extension</i> >> (p. 795) Controls whether or not a DDS_DataWriter (p. 485) receives DDS_DataWriterListener::on_application_acknowledgment (p. 1648) notifications with an empty or invalid response.
DDS_RtpsReliableReaderProtocol_t (p. 1954)	app_ack_period (p. 1958)
	<< <i>extension</i> >> (p. 795) The period at which application-level acknowledgment messages are sent
DDS_RtpsReliableReaderProtocol_t (p. 1954)	min_app_ack_response_keep_duration (p. 1958)
	<< <i>extension</i> >> (p. 795) Minimum duration for which application-level acknowledgment response data is kept
DDS_RtpsReliableReaderProtocol_t (p. 1954)	samples_per_app_ack (p. 1959)
	<< <i>extension</i> >> (p. 795) The minimum number of samples acknowledged by one application-level acknowledgment message

5.51 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. 1901)	value (p. 1903)
	<< basic >> (p. 783) << extension >> (p. 795) Sequence of properties.

Advanced Settings

DDS_UserDataQosPolicy (p. 2086)	value (p. 2087)
	<< spec >> (p. 790) a sequence of octets
DDS_GroupDataQosPolicy (p. 1803)	value (p. 1804)
	<< spec >> (p. 790) a sequence of octets
DDS_TypeSupportQosPolicy (p. 2081)	plugin_data (p. 2082)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Value to pass into the type plugin's de-/serialization function
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	participant_user_data_max_length (p. 1746)
	<< extension >> (p. 795) Maximum length of user data in DDS_DomainParticipantQos (p. 1726) and DDS_ParticipantBuiltinTopicData (p. 1870)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	topic_data_max_length (p. 1746)
	<< extension >> (p. 795) Maximum length of topic data in DDS_TopicQos (p. 2042), DDS_TopicBuiltinTopicData (p. 2034), DDS_PublicationBuiltinTopicData (p. 1905) and DDS_SubscriptionBuiltinTopicData (p. 2012)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	publisher_group_data_max_length (p. 1746)
	<< extension >> (p. 795) Maximum length of group data in DDS_PublisherQos (p. 1918) and DDS_PublicationBuiltinTopicData (p. 1905)

DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	subscriber_group_data_max_length (p. 1747)
	<< <i>extension</i> >> (p. 795) Maximum length of group data in DDS_SubscriberQos (p. 2010) and DDS_↔SubscriptionBuiltinTopicData (p. 2012)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	writer_user_data_max_length (p. 1747)
	<< <i>extension</i> >> (p. 795) Maximum length of user data in DDS_DataWriterQos (p. 1666) and DDS_↔PublicationBuiltinTopicData (p. 1905)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	reader_user_data_max_length (p. 1747)
	<< <i>extension</i> >> (p. 795) Maximum length of user data in DDS_DataReaderQos (p. 1611) and DDS_↔SubscriptionBuiltinTopicData (p. 2012)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	participant_property_list_max_length (p. 1752)
	<< <i>extension</i> >> (p. 795) Maximum number of properties associated with the DDS_DomainParticipant (p. 77).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	participant_property_string_max_length (p. 1753)
	<< <i>extension</i> >> (p. 795) Maximum string length of the properties associated with the DDS_DomainParticipant (p. 77).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	writer_property_list_max_length (p. 1753)
	<< <i>extension</i> >> (p. 795) Maximum number of properties associated with a DDS_DataWriter (p. 485).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	writer_property_string_max_length (p. 1753)
	<< <i>extension</i> >> (p. 795) Maximum string length of the properties associated with a DDS_DataWriter (p. 485).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	reader_property_list_max_length (p. 1754)
	<< <i>extension</i> >> (p. 795) Maximum number of properties associated with a DDS_DataReader (p. 615).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	reader_property_string_max_length (p. 1754)
	<< <i>extension</i> >> (p. 795) Maximum string length of the properties associated with a DDS_DataReader (p. 615).

5.52 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. 1788)	autoenable_created_entities (p. 1789)
	<< <i>basic</i> >> (p. 783) << <i>spec</i> >> (p. 790) Specifies whether the entity acting as a factory automatically enables the instances it creates.

5.53 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. 1552)	enable (p. 1552)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) Specifies whether or not batching is enabled.
DDS_BatchQosPolicy (p. 1552)	max_data_bytes (p. 1553)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) The maximum cumulative length of all serialized samples in a batch.
DDS_BatchQosPolicy (p. 1552)	max_samples (p. 1553)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) The maximum number of samples in a batch.

Advanced Settings

DDS_AsynchronousPublisherQosPolicy (p. 1539)	disable_asynchronous_batch (p. 1541)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Disable asynchronous batch flushing.
DDS_AsynchronousPublisherQosPolicy (p. 1539)	asynchronous_batch_thread (p. 1541)

	<< extension >> (p. 795) Settings of the batch flushing thread.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	initial_batches (p. 1676)
	<< extension >> (p. 795) Represents the initial number of batches a DDS_DataWriter (p. 485) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_batches (p. 1677)
	<< extension >> (p. 795) Represents the maximum number of batches a DDS_DataWriter (p. 485) will manage.
DDS_BatchQosPolicy (p. 1552)	max_flush_delay (p. 1554)
	<< extension >> (p. 795) The maximum flush delay.
DDS_BatchQosPolicy (p. 1552)	source_timestamp_resolution (p. 1554)
	<< extension >> (p. 795) Batch source timestamp resolution.
DDS_BatchQosPolicy (p. 1552)	thread_safe_write (p. 1555)
	<< extension >> (p. 795) Determines whether or not the write operation is thread safe.

5.54 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. 2062)	value (p. 2064)
	<< spec >> (p. 790) 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. 1539)	disable_asynchronous_write (p. 1540)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Disable asynchronous publishing.
DDS_AsynchronousPublisherQosPolicy (p. 1539)	thread (p. 1540)

	<< <i>extension</i> >> (p. 795) Settings of the publishing thread.
DDS_AsymchronousPublisherQosPolicy (p. 1539)	disable_asynchronous_batch (p. 1541)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Disable asynchronous batch flushing.
DDS_AsymchronousPublisherQosPolicy (p. 1539)	asynchronous_batch_thread (p. 1541)
	<< <i>extension</i> >> (p. 795) Settings of the batch flushing thread.
DDS_AsymchronousPublisherQosPolicy (p. 1539)	disable_topic_query_publication (p. 1542)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Disable topic query publication.
DDS_AsymchronousPublisherQosPolicy (p. 1539)	topic_query_publication_thread (p. 1542)
	<< <i>extension</i> >> (p. 795) Settings of the DDS_Topic ↔ Query (p. 707) publication thread.
DDS_DiscoveryQosPolicy (p. 1715)	metatraffic_transport_priority (p. 1717)
	<< <i>extension</i> >> (p. 795) The transport priority to use for the Discovery meta-traffic.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	initial_concurrent_blocking_threads (p. 1675)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The initial number of threads that are allowed to concurrently block on write call on the same DDS_DataWriter (p. 485).
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_concurrent_blocking_threads (p. 1675)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The maximum number of threads that are allowed to concurrently block on write call on the same DDS_Data ↔ Writer (p. 485).
DDS_EventQosPolicy (p. 1793)	thread (p. 1793)
	<< <i>extension</i> >> (p. 795) Event thread QoS.
DDS_DatabaseQosPolicy (p. 1581)	thread (p. 1582)
	<< <i>extension</i> >> (p. 795) Database thread settings.
DDS_ReceiverPoolQosPolicy (p. 1934)	thread (p. 1935)
	<< <i>extension</i> >> (p. 795) Receiver pool thread(s).
DDS_BatchQosPolicy (p. 1552)	thread_safe_write (p. 1555)
	<< <i>extension</i> >> (p. 795) Determines whether or not the write operation is thread safe.
DDS_MonitoringEventDistributionSettings (p. 1840)	concurrency_level (p. 1841)

	<< extension >> (p. 795) Defines how concurrent the push of event metrics to RTI Monitoring Library 2.0 is.
DDS_MonitoringEventDistributionSettings (p. 1840)	thread (p. 1842)
	<< extension >> (p. 795) The settings of the event metric thread.
DDS_MonitoringPeriodicDistributionSettings (p. 1852)	thread (p. 1852)
	<< extension >> (p. 795) The settings of the periodic metric thread.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	concurrency_level (p. 1844)
	<< extension >> (p. 795) Defines how concurrent the push of log messages to RTI Monitoring Library 2.0 is.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	thread (p. 1846)
	<< extension >> (p. 795) The settings of the logging thread.

5.55 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 **DDS_ContentFilteredTopic** (p. 177), but data can also be filtered using **DDS_TopicQuery** (p. 707) to run a new filter over the data in a DataWriter's cache, or **DDS_QueryCondition** (p. 699) to read a subset of the data that is currently stored in the DataReader's cache.

See also

Queries and Filters Syntax (p. 738).

Advanced Settings

DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_query_condition_filters (p. 1632)
	<< extension >> (p. 795) The maximum number of query condition filters a reader is allowed.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_remote_reader_filters (p. 1676)
	<< extension >> (p. 795) The maximum number of remote DataReaders for which the DDS_DataWriter (p. 485) will perform content-based filtering.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	content_filtered_topic_allocation (p. 1739)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to content filtered topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	content_filter_allocation (p. 1739)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to content filter.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	query_condition_allocation (p. 1740)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to query condition pool.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	content_filtered_topic_hash_buckets (p. 1744)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for content filtered topics.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	content_filter_hash_buckets (p. 1745)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for content filters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	contentfilter_property_max_length (p. 1751)
	<< <i>extension</i> >> (p. 795) This field is the maximum length of all data related to a Content-filtered topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	channel_seq_max_length (p. 1752)
	<< <i>extension</i> >> (p. 795) Maximum number of channels that can be specified in DDS_MultiChannelQosPolicy (p. 1856) for MultiChannel DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	channel_filter_expression_max_length (p. 1752)
	<< <i>extension</i> >> (p. 795) Maximum length of a channel DDS_ChannelSettings_t::filter_expression (p. 1564) in a MultiChannel DataWriter.
DDS_ChannelSettings_t (p. 1564)	multicast_settings (p. 1564)
	<< <i>extension</i> >> (p. 795) A sequence of DDS_TransportMulticastSettings_t (p. 2060) used to configure the multicast addresses associated with a channel.
DDS_ChannelSettings_t (p. 1564)	filter_expression (p. 1564)
	<< <i>extension</i> >> (p. 795) A logical expression used to determine the data that will be published in the channel.

DDS_MultiChannelQosPolicy (p. 1856)	filter_name (p. 1858)
	<< extension >> (p. 795) Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.

5.56 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. 1759)	kind (p. 1762)
	<< basic >> (p. 783) << spec >> (p. 790) The kind of durability.
DDS_DurabilityQosPolicy (p. 1759)	writer_depth (p. 1762)
	<< basic >> (p. 783) << extension >> (p. 795) Indicates the number of samples per instance that a durable DDS_DataWriter (p. 485) will send to a late-joining DDS_DataReader (p. 615).
DDS_DeadlineQosPolicy (p. 1684)	period (p. 1685)
	<< basic >> (p. 783) << spec >> (p. 790) Duration of the deadline period.
DDS_OwnershipQosPolicy (p. 1863)	kind (p. 1868)
	<< basic >> (p. 783) << spec >> (p. 790) The kind of ownership
DDS_OwnershipStrengthQosPolicy (p. 1868)	value (p. 1869)
	<< basic >> (p. 783) << spec >> (p. 790) The strength value used to arbitrate among multiple writers.
DDS_TimeBasedFilterQosPolicy (p. 2032)	minimum_separation (p. 2034)
	<< basic >> (p. 783) << spec >> (p. 790) The minimum separation duration between subsequent samples.
DDS_HistoryQosPolicy (p. 1805)	kind (p. 1807)
	<< basic >> (p. 783) << spec >> (p. 790) Specifies the kind of history to be kept.
DDS_HistoryQosPolicy (p. 1805)	depth (p. 1807)

	<< basic >> (p. 783) << spec >> (p. 790) Specifies the number of samples per instance to be kept, when the kind is DDS_KEEP_LAST_HISTORY_QOS (p. 1293)
DDS_ResourceLimitsQosPolicy (p. 1949)	max_samples (p. 1951)
	<< basic >> (p. 783) << spec >> (p. 790) Represents the maximum samples the middleware can store for any one DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)).
DDS_ResourceLimitsQosPolicy (p. 1949)	max_instances (p. 1951)
	<< basic >> (p. 783) << spec >> (p. 790) Represents the maximum number of instances a DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)) can manage.
DDS_LifespanQosPolicy (p. 1815)	duration (p. 1816)
	<< basic >> (p. 783) << spec >> (p. 790) Maximum duration for the data's validity.
DDS_MonitoringEventDistributionSettings (p. 1840)	publication_period (p. 1843)
	<< basic >> (p. 783) << extension >> (p. 795) Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.
DDS_MonitoringPeriodicDistributionSettings (p. 1852)	polling_period (p. 1853)
	<< basic >> (p. 783) << extension >> (p. 795) Period at which the periodic metric thread polls and publishes the periodic metrics.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	publication_period (p. 1846)
	<< basic >> (p. 783) << extension >> (p. 795) Period at which the logging thread publishes log messages.

Advanced Settings

DDS_AvailabilityQosPolicy (p. 1547)	enable_required_subscriptions (p. 1550)
	<< extension >> (p. 795) Enables support for required subscriptions in a DDS_DataWriter (p. 485)
DDS_AvailabilityQosPolicy (p. 1547)	max_data_availability_waiting_time (p. 1550)
	<< extension >> (p. 795) Defines how much time to wait before delivering a sample to the application without having received some of the previous samples.
DDS_AvailabilityQosPolicy (p. 1547)	max_endpoint_availability_waiting_time (p. 1550)

	<< extension >> (p. 795) Defines how much time to wait to discover DataWriters providing samples for the same data source (virtual GUID).
DDS_AvailabilityQosPolicy (p. 1547)	required_matched_endpoint_groups (p. 1551)
	<< extension >> (p. 795) A sequence of endpoint groups.
DDS_DurabilityQosPolicy (p. 1759)	direct_communication (p. 1762)
	<< extension >> (p. 795) Indicates whether or not a TRANSIENT or PERSISTENT DDS_DataReader (p. 615) should receive samples directly from a TRANSIENT or PERSISTENT DDS_DataWriter (p. 485)
DDS_DurabilityQosPolicy (p. 1759)	storage_settings (p. 1763)
	<< extension >> (p. 795) Configures durable writer history and durable reader state.
DDS_DurabilityServiceQosPolicy (p. 1764)	service_cleanup_delay (p. 1765)
	<< spec >> (p. 790) Controls when the service is able to remove all information regarding a data instances.
DDS_DurabilityServiceQosPolicy (p. 1764)	max_samples (p. 1766)
	<< spec >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1764)	max_instances (p. 1766)
	<< spec >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1764)	max_samples_per_instance (p. 1766)
	<< spec >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_PresentationQosPolicy (p. 1890)	access_scope (p. 1893)
	<< spec >> (p. 790) Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
DDS_PresentationQosPolicy (p. 1890)	coherent_access (p. 1893)
	<< spec >> (p. 790) Specifies support for <i>coherent</i> access. Controls whether coherent access is supported within the scope <code>access_scope</code> .
DDS_PresentationQosPolicy (p. 1890)	ordered_access (p. 1893)
	<< spec >> (p. 790) Specifies support for <i>ordered</i> access to the samples received at the subscription end. Controls whether ordered access is supported within the scope <code>access_scope</code> .

DDS_PresentationQosPolicy (p. 1890)	drop_incomplete_coherent_set (p. 1894)
	<< <i>extension</i> >> (p. 795) Indicates whether or not a DDS_DataReader (p. 615) 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. 1813)	duration (p. 1815)
	<< <i>spec</i> >> (p. 790) Duration of the maximum acceptable delay.
DDS_ResourceLimitsQosPolicy (p. 1949)	max_samples_per_instance (p. 1952)
	<< <i>spec</i> >> (p. 790) Represents the maximum number of samples of any one instance a DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)) can manage.
DDS_ReaderDataLifecycleQosPolicy (p. 1930)	autopurge_nowriter_samples_delay (p. 1932)
	<< <i>spec</i> >> (p. 790) Minimum duration for which the DDS_DataReader (p. 615) will maintain samples for an instance once its <i>instance_state</i> becomes DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE (p. 715).
DDS_ReaderDataLifecycleQosPolicy (p. 1930)	autopurge_disposed_samples_delay (p. 1932)
	<< <i>spec</i> >> (p. 790) Minimum duration for which the DDS_DataReader (p. 615) will maintain samples for an instance once its <i>instance_state</i> becomes DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE (p. 715).
DDS_ReaderDataLifecycleQosPolicy (p. 1930)	autopurge_disposed_instances_delay (p. 1932)
	<< <i>extension</i> >> (p. 795) Minimum duration for which the DDS_DataReader (p. 615) will maintain information about a received instance once its <i>instance_state</i> becomes DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE (p. 715) and there are no samples for the instance in the DataReader queue.
DDS_ReaderDataLifecycleQosPolicy (p. 1930)	autopurge_nowriter_instances_delay (p. 1933)
	<< <i>extension</i> >> (p. 795) Minimum duration for which the DDS_DataReader (p. 615) will maintain information about a received instance once its <i>instance_state</i> becomes DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE (p. 715) and there are no samples for the instance in the DataReader queue.
DDS_TransportMulticastMappingQosPolicy (p. 2056)	value (p. 2058)
	<< <i>extension</i> >> (p. 795) A sequence of multicast communication mappings.
DDS_TransportMulticastQosPolicy (p. 2058)	kind (p. 2059)

	<< extension >> (p. 795) A value that specifies a way to determine how to obtain the multicast address.
DDS_DataWriterProtocolQosPolicy (p. 1649)	virtual_guid (p. 1650)
	<< extension >> (p. 795) The virtual GUID (Global Unique Identifier).
DDS_DataWriterProtocolQosPolicy (p. 1649)	push_on_write (p. 1650)
	<< extension >> (p. 795) Whether to push sample out when write is called.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	initial_outstanding_reads (p. 1559)
	<< extension >> (p. 795) The initial number of outstanding reads that have not called finish yet on the same built-in topic DDS_DataReader (p. 615).
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_outstanding_reads (p. 1560)
	<< extension >> (p. 795) The maximum number of outstanding reads that have not called finish yet on the same built-in topic DDS_DataReader (p. 615).
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_samples_per_read (p. 1560)
	<< extension >> (p. 795) Maximum number of samples that can be read/taken on a same built-in topic DDS_DataReader (p. 615).
DDS_BatchQosPolicy (p. 1552)	source_timestamp_resolution (p. 1554)
	<< extension >> (p. 795) Batch source timestamp resolution.
DDS_ChannelSettings_t (p. 1564)	priority (p. 1565)
	<< extension >> (p. 795) Publication priority
DDS_MultiChannelQosPolicy (p. 1856)	channels (p. 1858)
	<< extension >> (p. 795) A sequence of DDS_ChannelSettings_t (p. 1564) used to configure the channels' properties. If the length of the sequence is zero, the QoS policy will be ignored.
DDS_MultiChannelQosPolicy (p. 1856)	filter_name (p. 1858)
	<< extension >> (p. 795) Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.

5.57 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 Connex applications.

Basic Settings

DDS_EntityNameQosPolicy (p. 1790)	name (p. 1790)
	<< basic >> (p. 783) << extension >> (p. 795) The name of the entity.
DDS_LoggingQosPolicy (p. 1831)	verbosity (p. 1832)
	<< basic >> (p. 783) << extension >> (p. 795) The verboisities at which RTI Connex diagnostic information is logged.
DDS_LoggingQosPolicy (p. 1831)	category (p. 1832)
	<< basic >> (p. 783) << extension >> (p. 795) Categories of logged messages.
DDS_LoggingQosPolicy (p. 1831)	output_file (p. 1832)
	<< basic >> (p. 783) << extension >> (p. 795) Specifies the file to which log messages will be redirected to.
DDS_LoggingQosPolicy (p. 1831)	output_file_suffix (p. 1833)
	<< basic >> (p. 783) << extension >> (p. 795) Sets the file suffix when logging to a set of files
DDS_MonitoringEventDistributionSettings (p. 1840)	publication_period (p. 1843)
	<< basic >> (p. 783) << extension >> (p. 795) Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.
DDS_MonitoringPeriodicDistributionSettings (p. 1852)	polling_period (p. 1853)
	<< basic >> (p. 783) << extension >> (p. 795) Period at which the periodic metric thread polls and publishes the periodic metrics.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	max_historical_logs (p. 1845)
	<< basic >> (p. 783) << extension >> (p. 795) The number of log messages that RTI Monitoring Library 2.0 will keep as history.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	publication_period (p. 1846)
	<< basic >> (p. 783) << extension >> (p. 795) Period at which the logging thread publishes log messages.
DDS_MonitoringDedicatedParticipantSettings (p. 1836)	enable (p. 1836)

	<< basic >> (p. 783) << extension >> (p. 795) Enables the use of a dedicated DDS_DomainParticipant (p. 77) to distribute the RTI Connex application telemetry data.
DDS_MonitoringDedicatedParticipantSettings (p. 1836)	domain_id (p. 1837)
	<< basic >> (p. 783) << extension >> (p. 795) The domain ID used in the creation of RTI Monitoring Library 2.0 DDS_DomainParticipant (p. 77).
DDS_MonitoringDedicatedParticipantSettings (p. 1836)	collector_initial_peers (p. 1837)
	<< basic >> (p. 783) << extension >> (p. 795) 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. 1839).
DDS_MonitoringLoggingForwardingSettings (p. 1847)	middleware_forwarding_level (p. 1847)
	<< basic >> (p. 783) << extension >> (p. 795) Log messages with NDDS_CONFIG_LOG_FACILITY ↵ MIDDLEWARE (p. 1447) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1847)	security_event_forwarding_level (p. 1847)
	<< basic >> (p. 783) << extension >> (p. 795) Log messages with NDDS_CONFIG_LOG_FACILITY ↵ SECURITY_EVENT (p. 1447) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1847)	service_forwarding_level (p. 1848)
	<< basic >> (p. 783) << extension >> (p. 795) Log messages with NDDS_CONFIG_LOG_FACILITY ↵ SERVICE (p. 1447) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1847)	user_forwarding_level (p. 1848)
	<< basic >> (p. 783) << extension >> (p. 795) Log messages with NDDS_CONFIG_LOG_FACILITY ↵ USER (p. 1447) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringTelemetryData (p. 1855)	metrics (p. 1855)

	<< basic >> (p. 783) << extension >> (p. 795) Sequence of DDS_MonitoringMetricSelection (p. 1849) containing the event and periodic metrics that will be collected and distributed for a given set of observable resources.
DDS_MonitoringTelemetryData (p. 1855)	logs (p. 1856)
	<< basic >> (p. 783) << extension >> (p. 795) DDS_MonitoringLoggingForwardingSettings (p. 1847) containing the NDDS_Config_Syslog_Verbosity (p. 1447) levels that will be forwarded for the different NDDS_Config_LogFacility (p. 1446).
DDS_MonitoringQosPolicy (p. 1853)	enable (p. 1854)
	<< basic >> (p. 783) << extension >> (p. 795) Enables the collection and distribution of telemetry data for an RTI Connext application using RTI Monitoring Library 2.0.
DDS_MonitoringQosPolicy (p. 1853)	application_name (p. 1854)
	<< basic >> (p. 783) << extension >> (p. 795) The name of the resource that represents this RTI Connext application.

Advanced Settings

DDS_DiscoveryConfigQosPolicy (p. 1689)	default_domain_announcement_period (p. 1704)
	<< extension >> (p. 795) The period to announce a participant to the default domain 0.
DDS_DiscoveryConfigQosPolicy (p. 1689)	ignore_default_domain_announcements (p. 1705)
	[DEPRECATED] (p. 965) << extension >> (p. 795) 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. 1790)	role_name (p. 1791)
	<< extension >> (p. 795) The entity role name.
DDS_ServiceQosPolicy (p. 2000)	kind (p. 2000)
	<< extension >> (p. 795) The kind of service.
DDS_LoggingQosPolicy (p. 1831)	print_format (p. 1832)
	<< extension >> (p. 795) The format used to output RTI Connext diagnostic information.
DDS_LoggingQosPolicy (p. 1831)	max_bytes_per_file (p. 1833)
	<< extension >> (p. 795) Specifies the maximum number of bytes a single file can contain

DDS_LoggingQosPolicy (p. 1831)	max_files (p. 1834)
	<< extension >> (p. 795) Specifies the maximum number of files to create before overwriting the previous ones.
DDS_MonitoringEventDistributionSettings (p. 1840)	concurrency_level (p. 1841)
	<< extension >> (p. 795) Defines how concurrent the push of event metrics to RTI Monitoring Library 2.0 is.
DDS_MonitoringEventDistributionSettings (p. 1840)	datawriter_qos_profile_name (p. 1842)
	<< extension >> (p. 795) The fully qualified name of the profile used to configure the DDS_DataWriter (p. 485) that distributes event metrics.
DDS_MonitoringEventDistributionSettings (p. 1840)	thread (p. 1842)
	<< extension >> (p. 795) The settings of the event metric thread.
DDS_MonitoringPeriodicDistributionSettings (p. 1852)	datawriter_qos_profile_name (p. 1852)
	<< extension >> (p. 795) The fully qualified name of the profile used to configure the DDS_DataWriter (p. 485) that distributes periodic metrics.
DDS_MonitoringPeriodicDistributionSettings (p. 1852)	thread (p. 1852)
	<< extension >> (p. 795) The settings of the periodic metric thread.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	concurrency_level (p. 1844)
	<< extension >> (p. 795) Defines how concurrent the push of log messages to RTI Monitoring Library 2.0 is.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	datawriter_qos_profile_name (p. 1845)
	<< extension >> (p. 795) The fully qualified name of the profile used to configure the DDS_DataWriter (p. 485) that distributes log messages.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	thread (p. 1846)
	<< extension >> (p. 795) The settings of the logging thread.
DDS_MonitoringDedicatedParticipantSettings (p. 1836)	participant_qos_profile_name (p. 1837)

	<< <i>extension</i> >> (p. 795) The fully qualified name of the profile used to configure the DDS_DomainParticipant (p. 77) that will be used to distribute telemetry data.
DDS_MonitoringDistributionSettings (p. 1838)	dedicated_participant (p. 1839)
	<< <i>extension</i> >> (p. 795) Configures the use of a dedicated DDS_DomainParticipant (p. 77) to distribute the RTI Connex application telemetry data.
DDS_MonitoringDistributionSettings (p. 1838)	publisher_qos_profile_name (p. 1839)
	<< <i>extension</i> >> (p. 795) The fully qualified name of the profile used to configure the Publishers that distribute telemetry data.
DDS_MonitoringDistributionSettings (p. 1838)	event_settings (p. 1839)
	<< <i>extension</i> >> (p. 795) Configures the distribution of event metrics.
DDS_MonitoringDistributionSettings (p. 1838)	periodic_settings (p. 1840)
	<< <i>extension</i> >> (p. 795) Configures the distribution of periodic metrics.
DDS_MonitoringDistributionSettings (p. 1838)	logging_settings (p. 1840)
	<< <i>extension</i> >> (p. 795) Configures the distribution of logging messages.
DDS_MonitoringMetricSelection (p. 1849)	resource_selection (p. 1849)
	<< <i>extension</i> >> (p. 795) An expression pattern that selects a subset of resources by matching the pattern to the resource names. DDS_MonitoringMetricSelection::enabled_metrics_selection (p. 1850) and DDS_MonitoringMetricSelection::disabled_metrics_selection (p. 1850) are applied to this subset of resources.
DDS_MonitoringMetricSelection (p. 1849)	enabled_metrics_selection (p. 1850)
	<< <i>extension</i> >> (p. 795) A sequence of POSIX fn-match 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. 1849).
DDS_MonitoringMetricSelection (p. 1849)	disabled_metrics_selection (p. 1850)
	<< <i>extension</i> >> (p. 795) A sequence of POSIX fn-match 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. 1849).
DDS_MonitoringQosPolicy (p. 1853)	distribution_settings (p. 1854)

	<< extension >> (p. 795) Configures the distribution of telemetry data.
DDS_MonitoringQosPolicy (p. 1853)	telemetry_data (p. 1855)
	<< extension >> (p. 795) Configures the telemetry data that will be distributed.

5.58 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. 2093)	participant_id (p. 2097)
	<< basic >> (p. 783) << extension >> (p. 795) A value used to distinguish among different participants belonging to the same domain on the same host.
DDS_DiscoveryQosPolicy (p. 1715)	initial_peers (p. 1716)
	<< basic >> (p. 783) << extension >> (p. 795) 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. 1715)	multicast_receive_addresses (p. 1717)
	<< basic >> (p. 783) << extension >> (p. 795) Specifies the multicast group addresses on which discovery-related meta-traffic can be received by the DomainParticipant.
DDS_DiscoveryQosPolicy (p. 1715)	accept_unknown_peers (p. 1718)
	<< basic >> (p. 783) << extension >> (p. 795) Whether to accept a new participant that is not in the initial peers list.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_liveliness_lease_duration (p. 1692)
	<< basic >> (p. 783) << extension >> (p. 795) The liveliness lease duration for the participant.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_liveliness_assert_period (p. 1692)
	<< basic >> (p. 783) << extension >> (p. 795) The period to assert liveliness for the participant.

DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_announcement_period (p. 1693)
	<< basic >> (p. 783) << extension >> (p. 795) The period at which a participant announces itself to potential peers when using the Simple Participant Discovery Protocol 2.0 (SPDP2).
DDS_DiscoveryConfigQosPolicy (p. 1689)	initial_participant_announcements (p. 1694)
	<< basic >> (p. 783) << extension >> (p. 795) The number of initial announcements sent when a participant is first enabled.
DDS_DiscoveryConfigQosPolicy (p. 1689)	new_remote_participant_announcements (p. 1695)
	<< basic >> (p. 783) << extension >> (p. 795) The number of participant announcements sent when a remote participant is newly discovered.
DDS_DiscoveryConfigQosPolicy (p. 1689)	min_initial_participant_announcement_period (p. 1695)
	<< basic >> (p. 783) << extension >> (p. 795) The minimum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
DDS_DiscoveryConfigQosPolicy (p. 1689)	max_initial_participant_announcement_period (p. 1695)
	<< basic >> (p. 783) << extension >> (p. 795) The maximum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
DDS_MonitoringDedicatedParticipantSettings (p. 1836)	domain_id (p. 1837)
	<< basic >> (p. 783) << extension >> (p. 795) The domain ID used in the creation of RTI Monitoring Library 2.0 DDS_DomainParticipant (p. 77).
DDS_MonitoringDedicatedParticipantSettings (p. 1836)	collector_initial_peers (p. 1837)
	<< basic >> (p. 783) << extension >> (p. 795) 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. 1839).

Advanced Settings

DDS_WireProtocolQosPolicy (p. 2093)	rtps_host_id (p. 2097)
	<< extension >> (p. 795) The RTPS Host ID of the domain participant.

DDS_WireProtocolQosPolicy (p. 2093)	rtps_app_id (p. 2098)
	<< <i>extension</i> >> (p. 795) The RTPS App ID of the domain participant.
DDS_WireProtocolQosPolicy (p. 2093)	rtps_instance_id (p. 2098)
	<< <i>extension</i> >> (p. 795) The RTPS Instance ID of the DDS_↵DomainParticipant (p. 77).
DDS_WireProtocolQosPolicy (p. 2093)	rtps_well_known_ports (p. 2099)
	<< <i>extension</i> >> (p. 795) Configures the RTPS well-known port mappings.
DDS_WireProtocolQosPolicy (p. 2093)	rtps_reserved_port_mask (p. 2099)
	<< <i>extension</i> >> (p. 795) Specifies which well-known ports to reserve when enabling the participant.
DDS_WireProtocolQosPolicy (p. 2093)	rtps_auto_id_kind (p. 2100)
	<< <i>extension</i> >> (p. 795) Kind of auto mechanism used to calculate the GUID prefix.
DDS_DiscoveryQosPolicy (p. 1715)	enabled_transports (p. 1716)
	<< <i>extension</i> >> (p. 795) The transports available for use by the Discovery mechanism.
DDS_DiscoveryQosPolicy (p. 1715)	metatraffic_transport_priority (p. 1717)
	<< <i>extension</i> >> (p. 795) The transport priority to use for the Discovery meta-traffic.
DDS_DiscoveryQosPolicy (p. 1715)	enable_endpoint_discovery (p. 1718)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Whether to automatically enable endpoint discovery for all the remote participants.
DDS_DiscoveryConfigQosPolicy (p. 1689)	remote_participant_purge_kind (p. 1693)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.
DDS_DiscoveryConfigQosPolicy (p. 1689)	max_liveliness_loss_detection_period (p. 1694)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) 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. 1689)	participant_reader_resource_limits (p. 1696)
	<< <i>extension</i> >> (p. 795) Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 1689)	publication_reader (p. 1696)

	<< extension >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in publication reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	publication_reader_resource_limits (p. 1697)
	<< extension >> (p. 795) Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 1689)	subscription_reader (p. 1697)
	<< extension >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in subscription reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	subscription_reader_resource_limits (p. 1698)
	<< extension >> (p. 795) Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_message_writer (p. 1702)
	<< extension >> (p. 795) 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. 1323) DDS_ReliabilityQos_↵PolicyKind (p. 1322).
DDS_DiscoveryConfigQosPolicy (p. 1689)	publication_writer_publish_mode (p. 1703)
	<< extension >> (p. 795) Publish mode policy for the built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	subscription_writer_publish_mode (p. 1704)
	<< extension >> (p. 795) Publish mode policy for the built-in subscription writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	asynchronous_publisher (p. 1704)
	<< extension >> (p. 795) Asynchronous publishing settings for the discovery DDS_Publisher (p. 443) and all entities that are created by it.
DDS_DiscoveryConfigQosPolicy (p. 1689)	default_domain_announcement_period (p. 1704)
	<< extension >> (p. 795) The period to announce a participant to the default domain 0.
DDS_DiscoveryConfigQosPolicy (p. 1689)	ignore_default_domain_announcements (p. 1705)
	[DEPRECATED] (p. 965) << extension >> (p. 795) 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. 1689)	participant_message_reader_reliability_kind (p. 1701)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Reliability policy for a built-in participant message reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_message_reader (p. 1702)

	<< extension >> (p. 795) 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. 1701) is set to DDS_RELIABLE_RELIABILITY_QOS (p. 1323).
DDS_DiscoveryConfigQosPolicy (p. 1689)	publication_writer (p. 1698)
	<< extension >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	publication_writer_data_lifecycle (p. 1699)
	<< extension >> (p. 795) Writer data lifecycle settings for a built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	subscription_writer (p. 1700)
	<< extension >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in subscription writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	subscription_writer_data_lifecycle (p. 1700)
	<< extension >> (p. 795) Writer data lifecycle settings for a built-in subscription writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	builtin_discovery_plugins (p. 1701)
	<< extension >> (p. 795) Mask of built-in discovery plugin kinds.
DDS_DiscoveryConfigQosPolicy (p. 1689)	service_request_writer (p. 1705)
	<< extension >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in DDS_ServiceRequest (p. 2001) writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	service_request_writer_data_lifecycle (p. 1706)
	<< extension >> (p. 795) Writer data lifecycle settings for a built-in DDS_ServiceRequest (p. 2001) writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	service_request_writer_publish_mode (p. 1706)
	<< extension >> (p. 795) Publish mode policy for the built-in service request writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	service_request_reader (p. 1707)
	<< extension >> (p. 795) RTPS reliable reader protocol-related configuration settings for a built-in DDS_ServiceRequest (p. 2001) reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	secure_volatile_writer (p. 1709)
	<< extension >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in secure volatile writer.

DDS_DiscoveryConfigQosPolicy (p. 1689)	secure_volatile_writer_publish_mode (p. 1710)
	<< <i>extension</i> >> (p. 795) Publish mode policy for the built-in secure volatile writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	secure_volatile_reader (p. 1710)
	<< <i>extension</i> >> (p. 795) RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	endpoint_type_object_lb_serialization_threshold (p. 1711)
	<< <i>extension</i> >> (p. 795) Option to reduce the size required to propagate a TypeObject in Simple Endpoint Discovery
DDS_DiscoveryConfigQosPolicy (p. 1689)	dns_tracker_polling_period (p. 1711)
	<< <i>extension</i> >> (p. 795) 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. 1689)	participant_configuration_reader (p. 1714)
	<< <i>extension</i> >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in participant configuration reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_configuration_reader_resource_limits (p. 1714)
	<< <i>extension</i> >> (p. 795) Resource limits for the built-in topic participant configuration reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_configuration_writer_publish_mode (p. 1712)
	<< <i>extension</i> >> (p. 795) Publish mode policy for the built-in participant configuration writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_configuration_writer (p. 1712)
	<< <i>extension</i> >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in participant configuration writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_configuration_writer_data_lifecycle (p. 1713)
	<< <i>extension</i> >> (p. 795) Writer data lifecycle settings for a built-in participant configuration writer.

5.59 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. 1881)	name (p. 1883)
	<< basic >> (p. 783) << spec >> (p. 790) A list of partition names
DDS_DiscoveryQosPolicy (p. 1715)	initial_peers (p. 1716)
	<< basic >> (p. 783) << extension >> (p. 795) 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. 1715)	accept_unknown_peers (p. 1718)
	<< basic >> (p. 783) << extension >> (p. 795) Whether to accept a new participant that is not in the initial peers list.
DDS_MonitoringDedicatedParticipantSettings (p. 1836)	domain_id (p. 1837)
	<< basic >> (p. 783) << extension >> (p. 795) The domain ID used in the creation of RTI Monitoring Library 2.0 DDS_DomainParticipant (p. 77).
DDS_MonitoringDedicatedParticipantSettings (p. 1836)	collector_initial_peers (p. 1837)
	<< basic >> (p. 783) << extension >> (p. 795) 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. 1839).

Advanced Settings

DDS_DiscoveryQosPolicy (p. 1715)	enable_endpoint_discovery (p. 1718)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Whether to automatically enable endpoint discovery for all the remote participants.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	max_partitions (p. 1748)
	<< extension >> (p. 795) Maximum number of partition name strings allowable in a DDS_PartitionQosPolicy (p. 1881).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	max_partition_cumulative_characters (p. 1748)
	<< extension >> (p. 795) Maximum number of combined characters allowable in all partition names in a DDS_PartitionQosPolicy (p. 1881).

5.60 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. 1759)	kind (p. 1762)
	<< <i>basic</i> >> (p. 783) << <i>spec</i> >> (p. 790) The kind of durability.
DDS_DurabilityQosPolicy (p. 1759)	writer_depth (p. 1762)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) Indicates the number of samples per instance that a durable DDS_DataWriter (p. 485) will send to a late-joining DDS_DataReader (p. 615).
DDS_ResourceLimitsQosPolicy (p. 1949)	max_samples (p. 1951)
	<< <i>basic</i> >> (p. 783) << <i>spec</i> >> (p. 790) Represents the maximum samples the middleware can store for any one DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)).
DDS_ResourceLimitsQosPolicy (p. 1949)	max_instances (p. 1951)
	<< <i>basic</i> >> (p. 783) << <i>spec</i> >> (p. 790) Represents the maximum number of instances a DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)) can manage.
DDS_LifespanQosPolicy (p. 1815)	duration (p. 1816)
	<< <i>basic</i> >> (p. 783) << <i>spec</i> >> (p. 790) Maximum duration for the data's validity.

Advanced Settings

DDS_AvailabilityQosPolicy (p. 1547)	enable_required_subscriptions (p. 1550)
	<< <i>extension</i> >> (p. 795) Enables support for required subscriptions in a DDS_DataWriter (p. 485)
DDS_AvailabilityQosPolicy (p. 1547)	max_data_availability_waiting_time (p. 1550)
	<< <i>extension</i> >> (p. 795) Defines how much time to wait before delivering a sample to the application without having received some of the previous samples.
DDS_AvailabilityQosPolicy (p. 1547)	max_endpoint_availability_waiting_time (p. 1550)
	<< <i>extension</i> >> (p. 795) Defines how much time to wait to discover DataWriters providing samples for the same data source (virtual GUID).

DDS_AvailabilityQosPolicy (p. 1547)	required_matched_endpoint_groups (p. 1551)
	<< <i>extension</i> >> (p. 795) A sequence of endpoint groups.
DDS_DurabilityQosPolicy (p. 1759)	direct_communication (p. 1762)
	<< <i>extension</i> >> (p. 795) Indicates whether or not a TRANSIENT or PERSISTENT DDS_DataReader (p. 615) should receive samples directly from a TRANSIENT or PERSISTENT DDS_DataWriter (p. 485)
DDS_DurabilityQosPolicy (p. 1759)	storage_settings (p. 1763)
	<< <i>extension</i> >> (p. 795) Configures durable writer history and durable reader state.
DDS_PersistentStorageSettings (p. 1883)	enable (p. 1885)
	<< <i>extension</i> >> (p. 795) Enables durable writer history in a DDS_DataWriter (p. 485) and durable reader state in a DDS_DataReader (p. 615).
DDS_PersistentStorageSettings (p. 1883)	file_name (p. 1885)
	<< <i>extension</i> >> (p. 795) The file name where the durable writer history or durable reader state will be stored.
DDS_PersistentStorageSettings (p. 1883)	trace_file_name (p. 1885)
	<< <i>extension</i> >> (p. 795) 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. 1883)	journal_kind (p. 1886)
	<< <i>extension</i> >> (p. 795) Sets the journal mode of the persistent storage.
DDS_PersistentStorageSettings (p. 1883)	synchronization_kind (p. 1886)
	<< <i>extension</i> >> (p. 795) Sets the level of synchronization with the physical disk.
DDS_PersistentStorageSettings (p. 1883)	vacuum (p. 1886)
	<< <i>extension</i> >> (p. 795) Sets the auto-vacuum status of the storage.
DDS_PersistentStorageSettings (p. 1883)	restore (p. 1887)
	<< <i>extension</i> >> (p. 795) Indicates if the persisted writer history or reader state must be restored.
DDS_PersistentStorageSettings (p. 1883)	writer_instance_cache_allocation (p. 1887)
	<< <i>extension</i> >> (p. 795) Configures the resource limits associated with the instance durable writer history cache.

DDS_PersistentStorageSettings (p. 1883)	writer_sample_cache_allocation (p. 1888)
	<< <i>extension</i> >> (p. 795) Configures the resource limits associated with the sample durable writer history cache.
DDS_PersistentStorageSettings (p. 1883)	writer_memory_state (p. 1889)
	<< <i>extension</i> >> (p. 795) 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. 1883)	reader_checkpoint_frequency (p. 1889)
	<< <i>extension</i> >> (p. 795) Controls how often the reader state is stored into the database.
DDS_DurabilityServiceQosPolicy (p. 1764)	service_cleanup_delay (p. 1765)
	<< <i>spec</i> >> (p. 790) Controls when the service is able to remove all information regarding a data instances.
DDS_DurabilityServiceQosPolicy (p. 1764)	history_kind (p. 1765)
	<< <i>spec</i> >> (p. 790) The kind of history to apply in re-couping durable data.
DDS_DurabilityServiceQosPolicy (p. 1764)	history_depth (p. 1765)
	<< <i>spec</i> >> (p. 790) Setting to use for the DDS_↔DurabilityQosPolicy::writer_depth (p. 1762) when re-couping durable data.
DDS_DurabilityServiceQosPolicy (p. 1764)	max_samples (p. 1766)
	<< <i>spec</i> >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1764)	max_instances (p. 1766)
	<< <i>spec</i> >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1764)	max_samples_per_instance (p. 1766)
	<< <i>spec</i> >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_ResourceLimitsQosPolicy (p. 1949)	max_samples_per_instance (p. 1952)
	<< <i>spec</i> >> (p. 790) Represents the maximum number of samples of any one instance a DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)) can manage.
DDS_DataWriterProtocolQosPolicy (p. 1649)	propagate_app_ack_with_no_response (p. 1653)
	<< <i>extension</i> >> (p. 795) Controls whether or not a DDS_DataWriter (p. 485) receives DDS_DataWriter↔Listener::on_application_acknowledgment (p. 1648) notifications with an empty or invalid response.

DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_enable_adaptive_sample_↔ keep_duration (p. 1971)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Enables dynamic adjustment of sample keep duration in response to congestion.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_min_sample_keep_duration (p. 1970)
	<< <i>extension</i> >> (p. 795) The minimum duration a sample is queued for ACK-disabled readers.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_max_sample_keep_duration (p. 1971)
	<< <i>extension</i> >> (p. 795) The maximum duration a sample is queued for ACK-disabled readers.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_decrease_sample_keep_↔ duration_factor (p. 1972)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Controls rate of contraction of dynamic sample keep duration.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_increase_sample_keep_↔ duration_factor (p. 1972)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Controls rate of growth of dynamic sample keep duration.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	initial_samples (p. 1558)
	<< <i>extension</i> >> (p. 795) Initial number of samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_samples (p. 1558)
	<< <i>extension</i> >> (p. 795) Maximum number of samples.
DDS_EntityNameQosPolicy (p. 1790)	role_name (p. 1791)
	<< <i>extension</i> >> (p. 795) The entity role name.

5.61 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. 1936)	max_blocking_time (p. 1939)
	<< basic >> (p. 783) << spec >> (p. 790) The maximum time a DataWriter can block on a write() call.
DDS_PublishModeQosPolicy (p. 1920)	kind (p. 1922)
	<< basic >> (p. 783) << extension >> (p. 795) Publishing mode
DDS_RtpsReliableWriterProtocol_t (p. 1960)	min_send_window_size (p. 1973)
	<< basic >> (p. 783) << extension >> (p. 795) Minimum size of send window of unacknowledged samples
DDS_RtpsReliableWriterProtocol_t (p. 1960)	max_send_window_size (p. 1973)
	<< basic >> (p. 783) << extension >> (p. 795) Maximum size of send window of unacknowledged samples

Advanced Settings

DDS_PublishModeQosPolicy (p. 1920)	flow_controller_name (p. 1922)
	<< extension >> (p. 795) Name of the associated flow controller
DDS_PublishModeQosPolicy (p. 1920)	priority (p. 1923)
	<< extension >> (p. 795) Publication priority
DDS_AsynchronousPublisherQosPolicy (p. 1539)	disable_asynchronous_write (p. 1540)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Disable asynchronous publishing.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	outstanding_asynchronous_sample_allocation (p. 1740)
	<< extension >> (p. 795) Allocation settings applied to the maximum number of samples (from all DDS_DataWriter (p. 485)) waiting to be asynchronously written.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	flow_controller_allocation (p. 1740)
	<< extension >> (p. 795) Allocation settings applied to flow controllers.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	flow_controller_hash_buckets (p. 1745)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Number of hash buckets for flow controllers.
DDS_DiscoveryConfigQosPolicy (p. 1689)	publication_writer_publish_mode (p. 1703)

	<< extension >> (p. 795) Publish mode policy for the built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	subscription_writer_publish_mode (p. 1704)
	<< extension >> (p. 795) Publish mode policy for the built-in subscription writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	asynchronous_publisher (p. 1704)
	<< extension >> (p. 795) Asynchronous publishing settings for the discovery DDS_Publisher (p. 443) and all entities that are created by it.
DDS_DiscoveryConfigQosPolicy (p. 1689)	service_request_writer_publish_mode (p. 1706)
	<< extension >> (p. 795) Publish mode policy for the built-in service request writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	secure_volatile_writer_publish_mode (p. 1710)
	<< extension >> (p. 795) Publish mode policy for the built-in secure volatile writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_configuration_writer_publish_mode (p. 1712)
	<< extension >> (p. 795) Publish mode policy for the built-in participant configuration writer.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	max_bytes_per_nack_response (p. 1969)
	<< extension >> (p. 795) The maximum total message size when resending rejected samples or data fragments.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	send_window_update_period (p. 1974)
	<< extension >> (p. 795) Period in which send window may be dynamically changed
DDS_RtpsReliableWriterProtocol_t (p. 1960)	send_window_increase_factor (p. 1975)
	<< extension >> (p. 795) Increases send window size by this percentage when reacting dynamically to network conditions
DDS_RtpsReliableWriterProtocol_t (p. 1960)	send_window_decrease_factor (p. 1976)
	<< extension >> (p. 795) Decreases send window size by this percentage when reacting dynamically to network conditions
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_repair_piggyback_heartbeat (p. 1978)
	<< extension >> (p. 795) Prevents piggyback heartbeats from being sent with repair samples

5.62 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. 1619)	disable_fragmentation_support (p. 1626)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Determines whether the DDS_DataReader (p. 615) can receive fragmented samples.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_fragmented_samples (p. 1626)
	<< <i>extension</i> >> (p. 795) The maximum number of samples for which the DDS_DataReader (p. 615) may store fragments at a given point in time.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_fragmented_samples (p. 1627)
	<< <i>extension</i> >> (p. 795) The initial number of samples for which a DDS_DataReader (p. 615) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_fragmented_samples_per_remote_writer (p. 1627)
	<< <i>extension</i> >> (p. 795) The maximum number of samples per remote writer for which a DDS_DataReader (p. 615) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_fragments_per_sample (p. 1628)
	<< <i>extension</i> >> (p. 795) Maximum number of fragments for a single sample.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	dynamically_allocate_fragmented_samples (p. 1628)
	<< <i>extension</i> >> (p. 795) Determines whether the DDS_DataReader (p. 615) pre-allocates storage for storing fragmented samples.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	min_nack_response_delay (p. 1968)
	<< <i>extension</i> >> (p. 795) The minimum delay to respond to a NACK or NACK_FRAG.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	max_nack_response_delay (p. 1968)
	<< <i>extension</i> >> (p. 795) The maximum delay to respond to a NACK or NACK_FRAG.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	nack_suppression_duration (p. 1969)

	<< extension >> (p. 795) The duration for ignoring consecutive NACKs/NACK_FRAGs that may trigger redundant repairs.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	max_bytes_per_nack_response (p. 1969)
	<< extension >> (p. 795) The maximum total message size when resending rejected samples or data fragments.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	disable_fragmentation_support (p. 1560)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Determines whether the built-in topic DDS_DataReader (p. 615) can receive fragmented samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_fragmented_samples (p. 1561)
	<< extension >> (p. 795) The maximum number of samples for which the built-in topic DDS_DataReader (p. 615) may store fragments at a given point in time.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	initial_fragmented_samples (p. 1561)
	<< extension >> (p. 795) The initial number of samples for which a built-in topic DDS_DataReader (p. 615) may store fragments.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_fragmented_samples_per_remote_writer (p. 1562)
	<< extension >> (p. 795) The maximum number of samples per remote writer for which a built-in topic DDS_DataReader (p. 615) may store fragments.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_fragments_per_sample (p. 1562)
	<< extension >> (p. 795) Maximum number of fragments for a single sample.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	dynamically_allocate_fragmented_samples (p. 1563)
	<< extension >> (p. 795) Determines whether the built-in topic DDS_DataReader (p. 615) pre-allocates storage for storing fragmented samples.

5.63 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. 1949)	max_instances (p. 1951)
	<< <i>basic</i> >> (p. 783) << <i>spec</i> >> (p. 790) Represents the maximum number of instances a DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)) can manage.
DDS_ResourceLimitsQosPolicy (p. 1949)	initial_instances (p. 1952)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) Represents the initial number of instances a DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)) will manage.

Advanced Settings

DDS_DurabilityServiceQosPolicy (p. 1764)	service_cleanup_delay (p. 1765)
	<< <i>spec</i> >> (p. 790) Controls when the service is able to remove all information regarding a data instances.
DDS_DurabilityServiceQosPolicy (p. 1764)	max_instances (p. 1766)
	<< <i>spec</i> >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1764)	max_samples_per_instance (p. 1766)
	<< <i>spec</i> >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_ReliabilityQosPolicy (p. 1936)	instance_state_consistency_kind (p. 1940)
	<< <i>extension</i> >> (p. 795) Whether instance state consistency is enabled
DDS_ResourceLimitsQosPolicy (p. 1949)	max_samples_per_instance (p. 1952)
	<< <i>spec</i> >> (p. 790) Represents the maximum number of samples of any one instance a DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)) can manage.
DDS_ResourceLimitsQosPolicy (p. 1949)	instance_hash_buckets (p. 1953)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for instances.
DDS_WriterDataLifecycleQosPolicy (p. 2106)	autodispose_unregistered_instances (p. 2107)
	<< <i>spec</i> >> (p. 790) Boolean flag that controls whether or not a writer will also dispose of an instance when the DDS_DataWriter (p. 485) unregisters from the instance by means of the unregister operations.
DDS_WriterDataLifecycleQosPolicy (p. 2106)	autopurge_unregistered_instances_delay (p. 2107)

	<< extension >> (p. 795) Maximum duration for which the DDS_DataWriter (p. 485) will maintain information regarding an instance once it has unregistered the instance.
DDS_WriterDataLifecycleQosPolicy (p. 2106)	autopurge_disposed_instances_delay (p. 2108)
	<< extension >> (p. 795) Maximum duration for which the DDS_DataWriter (p. 485) will maintain information regarding an instance once it has disposed the instance.
DDS_ReaderDataLifecycleQosPolicy (p. 1930)	autopurge_nowriter_samples_delay (p. 1932)
	<< spec >> (p. 790) Minimum duration for which the DDS_DataReader (p. 615) will maintain samples for an instance once its <code>instance_state</code> becomes DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE (p. 715).
DDS_ReaderDataLifecycleQosPolicy (p. 1930)	autopurge_disposed_samples_delay (p. 1932)
	<< spec >> (p. 790) Minimum duration for which the DDS_DataReader (p. 615) will maintain samples for an instance once its <code>instance_state</code> becomes DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE (p. 715).
DDS_ReaderDataLifecycleQosPolicy (p. 1930)	autopurge_disposed_instances_delay (p. 1932)
	<< extension >> (p. 795) Minimum duration for which the DDS_DataReader (p. 615) will maintain information about a received instance once its <code>instance_state</code> becomes DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE (p. 715) and there are no samples for the instance in the DataReader queue.
DDS_ReaderDataLifecycleQosPolicy (p. 1930)	autopurge_nowriter_instances_delay (p. 1933)
	<< extension >> (p. 795) Minimum duration for which the DDS_DataReader (p. 615) will maintain information about a received instance once its <code>instance_state</code> becomes DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE (p. 715) and there are no samples for the instance in the DataReader queue.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_total_instances (p. 1629)
	<< extension >> (p. 795) Maximum number of instances for which a DataReader will keep state.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	keep_minimum_state_for_instances (p. 1633)
	<< extension >> (p. 795) Whether or not keep a minimum instance state for up to DDS_DataReaderResourceLimitsQosPolicy::max_total_instances (p. 1629).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	instance_replacement (p. 1635)

	<< extension >> (p. 795) Sets the kind of instances allowed to be replaced for each instance state (DDS_↵_InstanceStateKind (p. 714)) when a DataReader reaches DDS_ResourceLimitsQosPolicy::max_↵instances (p. 1951).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	autopurge_remote_not_alive_writer_delay (p. 1636)
	<< extension >> (p. 795) Maximum duration for which the DDS_DataReader (p. 615) will maintain information regarding a DDS_DataWriter (p. 485) once the DDS_↵DataWriter (p. 485) has become not alive.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	instance_replacement (p. 1677)
	<< extension >> (p. 795) Sets the kinds of instances allowed to be replaced when instance resource limits are reached.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	replace_empty_instances (p. 1678)
	<< extension >> (p. 795) Whether or not to replace empty instances during instance replacement
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	autoregister_instances (p. 1678)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Whether or not to automatically register new instances
DDS_DataReaderProtocolQosPolicy (p. 1596)	propagate_dispose_of_unregistered_instances (p. 1598)
	<< extension >> (p. 795) Indicates whether or not an instance can move to the DDS_NOT_ALIVE_↵DISPOSED_INSTANCE_STATE (p. 715) state without being in the DDS_ALIVE_INSTANCE_STATE (p. 715) state.
DDS_DataReaderProtocolQosPolicy (p. 1596)	propagate_unregister_of_disposed_instances (p. 1599)
	<< extension >> (p. 795) Indicates whether or not an instance can move to the DDS_NOT_ALIVE_NO_↵_WRITERS_INSTANCE_STATE (p. 715) state directly from the DDS_NOT_ALIVE_DISPOSED_INSTANCE_↵_STATE (p. 715).
DDS_DataWriterProtocolQosPolicy (p. 1649)	disable_inline_keyhash (p. 1651)
	<< extension >> (p. 795) Controls whether or not a keyhash is propagated on the wire with each sample.
DDS_DataWriterProtocolQosPolicy (p. 1649)	serialize_key_with_dispose (p. 1652)
	<< extension >> (p. 795) Controls whether or not the serialized key is propagated on the wire with dispose samples.
DDS_DiscoveryConfigQosPolicy (p. 1689)	publication_writer_data_lifecycle (p. 1699)

	<< extension >> (p. 795) Writer data lifecycle settings for a built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	subscription_writer_data_lifecycle (p. 1700)
	<< extension >> (p. 795) Writer data lifecycle settings for a built-in subscription writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	service_request_writer_data_lifecycle (p. 1706)
	<< extension >> (p. 795) Writer data lifecycle settings for a built-in DDS_ServiceRequest (p. 2001) writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_configuration_writer_data_lifecycle (p. 1713)
	<< extension >> (p. 795) Writer data lifecycle settings for a built-in participant configuration writer.

5.64 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. 1638)	compression_settings (p. 1640)
	<< basic >> (p. 783) << extension >> (p. 795) Structure that contains the compression settings.

Advanced Settings

DDS_CompressionSettings_t (p. 1568)	compression_ids (p. 1568)
	<< extension >> (p. 795) Mask that represents the compression algorithms enabled.
DDS_CompressionSettings_t (p. 1568)	writer_compression_level (p. 1569)
	<< extension >> (p. 795) The level of compression to use when compressing data.
DDS_CompressionSettings_t (p. 1568)	writer_compression_threshold (p. 1569)

	<< <i>extension</i> >> (p. 795) The threshold, in bytes, above which a serialized sample will be eligible to be compressed.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	disable_fragmentation_support (p. 1626)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Determines whether the DDS_DataReader (p. 615) can receive fragmented samples.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_fragmented_samples (p. 1626)
	<< <i>extension</i> >> (p. 795) The maximum number of samples for which the DDS_DataReader (p. 615) may store fragments at a given point in time.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_fragmented_samples (p. 1627)
	<< <i>extension</i> >> (p. 795) The initial number of samples for which a DDS_DataReader (p. 615) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_fragmented_samples_per_remote_writer (p. 1627)
	<< <i>extension</i> >> (p. 795) The maximum number of samples per remote writer for which a DDS_DataReader (p. 615) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_fragments_per_sample (p. 1628)
	<< <i>extension</i> >> (p. 795) Maximum number of fragments for a single sample.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	dynamically_allocate_fragmented_samples (p. 1628)
	<< <i>extension</i> >> (p. 795) Determines whether the DDS_DataReader (p. 615) pre-allocates storage for storing fragmented samples.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	shmem_ref_transfer_mode_attached_segment_allocation (p. 1635)
	<< <i>extension</i> >> (p. 795) Allocation resource for the shared memory segments attached by the DDS_DataReader (p. 615).
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	writer_loaned_sample_allocation (p. 1681)
	<< <i>extension</i> >> (p. 795) Represents the allocation settings of loaned samples managed by a DDS_DataWriter (p. 485).
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	initialize_writer_loaned_sample (p. 1681)
	<< <i>extension</i> >> (p. 795) Whether or not to initialize loaned samples returned by a DDS_DataWriter (p. 485)
DDS_DataWriterTransferModeQosPolicy (p. 1683)	shmem_ref_settings (p. 1683)

	<< <i>extension</i> >> (p. 795) Settings related to transferring data using shared memory references.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	shmem_ref_transfer_mode_max_segments (p. 1758)
	<< <i>extension</i> >> (p. 795) Maximum number of segments created by all DataWriters belonging to a DDS_↵_DomainParticipant (p. 77).
DDS_DiscoveryConfigQosPolicy (p. 1689)	endpoint_type_object_lb_serialization_threshold (p. 1711)
	<< <i>extension</i> >> (p. 795) Option to reduce the size required to propagate a TypeObject in Simple Endpoint Discovery
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	disable_fragmentation_support (p. 1560)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Determines whether the built-in topic DDS_DataReader (p. 615) can receive fragmented samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_fragmented_samples (p. 1561)
	<< <i>extension</i> >> (p. 795) The maximum number of samples for which the built-in topic DDS_DataReader (p. 615) may store fragments at a given point in time.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	initial_fragmented_samples (p. 1561)
	<< <i>extension</i> >> (p. 795) The initial number of samples for which a built-in topic DDS_DataReader (p. 615) may store fragments.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_fragmented_samples_per_remote_writer (p. 1562)
	<< <i>extension</i> >> (p. 795) The maximum number of samples per remote writer for which a built-in topic DDS_DataReader (p. 615) may store fragments.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_fragments_per_sample (p. 1562)
	<< <i>extension</i> >> (p. 795) Maximum number of fragments for a single sample.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	dynamically_allocate_fragmented_samples (p. 1563)
	<< <i>extension</i> >> (p. 795) Determines whether the built-in topic DDS_DataReader (p. 615) pre-allocates storage for storing fragmented samples.

5.65 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. 1823)	kind (p. 1825)
	<< basic >> (p. 783) << spec >> (p. 790) The kind of liveliness desired.
DDS_LivelinessQosPolicy (p. 1823)	lease_duration (p. 1825)
	<< basic >> (p. 783) << spec >> (p. 790) The duration within which a DDS_DataWriter (p. 485) must be asserted, or else it is assumed to be not alive.
DDS_LivelinessQosPolicy (p. 1823)	assertions_per_lease_duration (p. 1826)
	<< basic >> (p. 783) << extension >> (p. 795) The number of assertions a DDS_DataWriter (p. 485) will send during a its DDS_↔LivelinessQosPolicy::lease_duration (p. 1825).
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_liveliness_lease_duration (p. 1692)
	<< basic >> (p. 783) << extension >> (p. 795) The liveliness lease duration for the participant.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_liveliness_assert_period (p. 1692)
	<< basic >> (p. 783) << extension >> (p. 795) The period to assert liveliness for the participant.

Advanced Settings

DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	autopurge_remote_not_alive_writer_delay (p. 1636)
	<< extension >> (p. 795) Maximum duration for which the DDS_DataReader (p. 615) will maintain information regarding a DDS_DataWriter (p. 485) once the DDS_↔DataWriter (p. 485) has become not alive.
DDS_DiscoveryConfigQosPolicy (p. 1689)	remote_participant_purge_kind (p. 1693)
	[DEPRECATED] (p. 965) << extension >> (p. 795) The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.
DDS_DiscoveryConfigQosPolicy (p. 1689)	max_liveliness_loss_detection_period (p. 1694)

	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) 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. 1689)	participant_message_writer (p. 1702)
	<< <i>extension</i> >> (p. 795) 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. 1323) DDS_ReliabilityQos_↵PolicyKind (p. 1322).
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_message_reader_reliability_kind (p. 1701)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Reliability policy for a built-in participant message reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_message_reader (p. 1702)
	<< <i>extension</i> >> (p. 795) 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. 1701) is set to DDS_RELIABLE_RELIABILITY_QOS (p. 1323).
DDS_RtpsReliableWriterProtocol_t (p. 1960)	max_heartbeat_retries (p. 1966)
	<< <i>extension</i> >> (p. 795) The maximum number of <i>periodic</i> heartbeat retries before marking a remote reader as inactive.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	inactivate_nonprogressing_readers (p. 1966)
	<< <i>extension</i> >> (p. 795) Whether to treat remote readers as inactive when their NACKs do not progress

5.66 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. 1831)	verbosity (p. 1832)
	<< basic >> (p. 783) << extension >> (p. 795) The verbosity at which RTI Connex diagnostic information is logged.
DDS_LoggingQosPolicy (p. 1831)	category (p. 1832)
	<< basic >> (p. 783) << extension >> (p. 795) Categories of logged messages.
DDS_LoggingQosPolicy (p. 1831)	output_file (p. 1832)
	<< basic >> (p. 783) << extension >> (p. 795) Specifies the file to which log messages will be redirected to.
DDS_LoggingQosPolicy (p. 1831)	output_file_suffix (p. 1833)
	<< basic >> (p. 783) << extension >> (p. 795) Sets the file suffix when logging to a set of files
DDS_MonitoringLoggingDistributionSettings (p. 1843)	max_historical_logs (p. 1845)
	<< basic >> (p. 783) << extension >> (p. 795) The number of log messages that RTI Monitoring Library 2.0 will keep as history.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	publication_period (p. 1846)
	<< basic >> (p. 783) << extension >> (p. 795) Period at which the logging thread publishes log messages.
DDS_MonitoringLoggingForwardingSettings (p. 1847)	middleware_forwarding_level (p. 1847)
	<< basic >> (p. 783) << extension >> (p. 795) Log messages with NDDS_CONFIG_LOG_FACILITY_↔ MIDDLEWARE (p. 1447) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1847)	security_event_forwarding_level (p. 1847)
	<< basic >> (p. 783) << extension >> (p. 795) Log messages with NDDS_CONFIG_LOG_FACILITY_↔ SECURITY_EVENT (p. 1447) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1847)	service_forwarding_level (p. 1848)

	<< basic >> (p. 783) << extension >> (p. 795) Log messages with NDDS_CONFIG_LOG_FACILITY_↵SERVICE (p. 1447) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1847)	user_forwarding_level (p. 1848)
	<< basic >> (p. 783) << extension >> (p. 795) Log messages with NDDS_CONFIG_LOG_FACILITY_↵USER (p. 1447) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringTelemetryData (p. 1855)	logs (p. 1856)
	<< basic >> (p. 783) << extension >> (p. 795) DDS_MonitoringLoggingForwardingSettings (p. 1847) containing the NDDS_Config_Syslog_↵Verbosity (p. 1447) levels that will be forwarded for the different NDDS_Config_LogFacility (p. 1446).

Advanced Settings

DDS_LoggingQosPolicy (p. 1831)	print_format (p. 1832)
	<< extension >> (p. 795) The format used to output RTI Connex diagnostic information.
DDS_LoggingQosPolicy (p. 1831)	max_bytes_per_file (p. 1833)
	<< extension >> (p. 795) Specifies the maximum number of bytes a single file can contain
DDS_LoggingQosPolicy (p. 1831)	max_files (p. 1834)
	<< extension >> (p. 795) Specifies the maximum number of files to create before overwriting the previous ones.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	concurrency_level (p. 1844)
	<< extension >> (p. 795) Defines how concurrent the push of log messages to RTI Monitoring Library 2.0 is.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	datawriter_qos_profile_name (p. 1845)
	<< extension >> (p. 795) The fully qualified name of the profile used to configure the DDS_DataWriter (p. 485) that distributes log messages.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	thread (p. 1846)

	<< extension >> (p. 795) The settings of the logging thread.
DDS_MonitoringDistributionSettings (p. 1838)	logging_settings (p. 1840)
	<< extension >> (p. 795) Configures the distribution of logging messages.

5.67 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. 1949)	max_samples (p. 1951)
	<< basic >> (p. 783) << spec >> (p. 790) Represents the maximum samples the middleware can store for any one DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)).
DDS_ResourceLimitsQosPolicy (p. 1949)	max_instances (p. 1951)
	<< basic >> (p. 783) << spec >> (p. 790) Represents the maximum number of instances a DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)) can manage.
DDS_ResourceLimitsQosPolicy (p. 1949)	initial_samples (p. 1952)
	<< basic >> (p. 783) << extension >> (p. 795) Represents the initial samples the middleware will store for any one DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)).
DDS_ResourceLimitsQosPolicy (p. 1949)	initial_instances (p. 1952)
	<< basic >> (p. 783) << extension >> (p. 795) Represents the initial number of instances a DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)) will manage.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	max_historical_logs (p. 1845)
	<< basic >> (p. 783) << extension >> (p. 795) The number of log messages that RTI Monitoring Library 2.0 will keep as history.

Advanced Settings

DDS_PersistentStorageSettings (p. 1883)	writer_memory_state (p. 1889)
	<< <i>extension</i> >> (p. 795) 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. 1764)	max_samples (p. 1766)
	<< <i>spec</i> >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1764)	max_instances (p. 1766)
	<< <i>spec</i> >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1764)	max_samples_per_instance (p. 1766)
	<< <i>spec</i> >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_ResourceLimitsQosPolicy (p. 1949)	max_samples_per_instance (p. 1952)
	<< <i>spec</i> >> (p. 790) Represents the maximum number of samples of any one instance a DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)) can manage.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_remote_writers (p. 1621)
	<< <i>extension</i> >> (p. 795) The maximum number of remote writers from which a DDS_DataReader (p. 615) may read, including all instances.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_remote_writers_per_instance (p. 1622)
	<< <i>extension</i> >> (p. 795) The maximum number of remote writers from which a DDS_DataReader (p. 615) may read a single instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_samples_per_remote_writer (p. 1622)
	<< <i>extension</i> >> (p. 795) The maximum number of out-of-order samples from a given remote DDS_DataWriter (p. 485) that a DDS_DataReader (p. 615) may store when maintaining a reliable connection to the DDS_↔DataWriter (p. 485).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_infos (p. 1623)
	<< <i>extension</i> >> (p. 795) The maximum number of info units that a DDS_DataReader (p. 615) can use to store DDS_SampleInfo (p. 1985).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_remote_writers (p. 1623)
	<< <i>extension</i> >> (p. 795) The initial number of remote writers from which a DDS_DataReader (p. 615) may read, including all instances.

DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_remote_writers_per_instance (p. 1624)
	<< <i>extension</i> >> (p. 795) The initial number of remote writers from which a DDS_DataReader (p. 615) may read a single instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_infos (p. 1624)
	<< <i>extension</i> >> (p. 795) The initial number of info units that a DDS_DataReader (p. 615) can have, which are used to store DDS_SampleInfo (p. 1985).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_outstanding_reads (p. 1625)
	<< <i>extension</i> >> (p. 795) The initial number of outstanding calls to read/take (or one of their variants) on the same DDS_DataReader (p. 615) for which memory has not been returned by calling FooDataReader_↔return_loan (p. 646).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_outstanding_reads (p. 1625)
	<< <i>extension</i> >> (p. 795) The maximum number of outstanding read/take calls (or one of their variants) on the same DDS_DataReader (p. 615) for which memory has not been returned by calling FooDataReader_return_↔_loan (p. 646).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_samples_per_read (p. 1625)
	<< <i>extension</i> >> (p. 795) The maximum number of data samples that the application can receive from the middleware in a single call to FooDataReader_read (p. 625) or FooDataReader_take (p. 626). If more data exists in the middleware, the application will need to issue multiple read/take calls.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_fragmented_samples (p. 1626)
	<< <i>extension</i> >> (p. 795) The maximum number of samples for which the DDS_DataReader (p. 615) may store fragments at a given point in time.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_fragmented_samples (p. 1627)
	<< <i>extension</i> >> (p. 795) The initial number of samples for which a DDS_DataReader (p. 615) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_fragmented_samples_per_remote_writer (p. 1627)
	<< <i>extension</i> >> (p. 795) The maximum number of samples per remote writer for which a DDS_DataReader (p. 615) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_fragments_per_sample (p. 1628)

	<< <i>extension</i> >> (p. 795) Maximum number of fragments for a single sample.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	dynamically_allocate_fragmented_samples (p. 1628)
	<< <i>extension</i> >> (p. 795) Determines whether the DDS_DataReader (p. 615) pre-allocates storage for storing fragmented samples.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_total_instances (p. 1629)
	<< <i>extension</i> >> (p. 795) Maximum number of instances for which a DataReader will keep state.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_remote_virtual_writers (p. 1630)
	<< <i>extension</i> >> (p. 795) The maximum number of remote virtual writers from which a DDS_DataReader (p. 615) may read, including all instances.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_remote_virtual_writers (p. 1630)
	<< <i>extension</i> >> (p. 795) The initial number of remote virtual writers from which a DDS_DataReader (p. 615) may read, including all instances.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_remote_virtual_writers_per_instance (p. 1631)
	<< <i>extension</i> >> (p. 795) The maximum number of virtual remote writers that can be associated with an instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_remote_virtual_writers_per_instance (p. 1631)
	<< <i>extension</i> >> (p. 795) The initial number of virtual remote writers per instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_remote_writers_per_sample (p. 1632)
	<< <i>extension</i> >> (p. 795) The maximum number of remote writers allowed to write the same sample.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_query_condition_filters (p. 1632)
	<< <i>extension</i> >> (p. 795) The maximum number of query condition filters a reader is allowed.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_app_ack_response_length (p. 1633)
	<< <i>extension</i> >> (p. 795) Maximum length of application-level acknowledgment response data
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	keep_minimum_state_for_instances (p. 1633)
	<< <i>extension</i> >> (p. 795) Whether or not keep a minimum instance state for up to DDS_DataReaderResourceLimitsQosPolicy::max_total_instances (p. 1629).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_topic_queries (p. 1634)

	<< <i>extension</i> >> (p. 795) The initial number of Topic↔ Queries allocated by a DDS_DataReader (p. 615).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_topic_queries (p. 1634)
	<< <i>extension</i> >> (p. 795) The maximum number of active TopicQueries that a DDS_DataReader (p. 615) can create.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	shmem_ref_transfer_mode_attached_segment_↔ allocation (p. 1635)
	<< <i>extension</i> >> (p. 795) Allocation resource for the shared memory segments attached by the DDS_Data↔ Reader (p. 615).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	instance_replacement (p. 1635)
	<< <i>extension</i> >> (p. 795) Sets the kind of instances allowed to be replaced for each instance state (DDS_↔ _InstanceStateKind (p. 714)) when a DataReader reaches DDS_ResourceLimitsQosPolicy::max_↔ instances (p. 1951).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	autopurge_remote_not_alive_writer_delay (p. 1636)
	<< <i>extension</i> >> (p. 795) Maximum duration for which the DDS_DataReader (p. 615) will maintain information regarding a DDS_DataWriter (p. 485) once the DDS_↔ DataWriter (p. 485) has become not alive.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	initial_concurrent_blocking_threads (p. 1675)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The initial number of threads that are allowed to concurrently block on write call on the same DDS_DataWriter (p. 485).
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_concurrent_blocking_threads (p. 1675)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The maximum number of threads that are allowed to concurrently block on write call on the same DDS_Data↔ Writer (p. 485).
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_remote_reader_filters (p. 1676)
	<< <i>extension</i> >> (p. 795) The maximum number of remote DataReaders for which the DDS_DataWriter (p. 485) will perform content-based filtering.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	initial_batches (p. 1676)
	<< <i>extension</i> >> (p. 795) Represents the initial number of batches a DDS_DataWriter (p. 485) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_batches (p. 1677)

	<< extension >> (p. 795) Represents the maximum number of batches a DDS_DataWriter (p. 485) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	instance_replacement (p. 1677)
	<< extension >> (p. 795) Sets the kinds of instances allowed to be replaced when instance resource limits are reached.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	replace_empty_instances (p. 1678)
	<< extension >> (p. 795) Whether or not to replace empty instances during instance replacement
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_virtual_writers (p. 1679)
	<< extension >> (p. 795) The maximum number of virtual writers supported by a DDS_DataWriter (p. 485)
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	initial_virtual_writers (p. 1679)
	<< extension >> (p. 795) The initial number of virtual writers supported by a DDS_DataWriter (p. 485)
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_remote_readers (p. 1679)
	<< extension >> (p. 795) The maximum number of remote readers supported by a DDS_DataWriter (p. 485)
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_app_ack_remote_readers (p. 1680)
	<< extension >> (p. 795) The maximum number of application-level acknowledging remote readers supported by a DDS_DataWriter (p. 485)
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	initial_active_topic_queries (p. 1680)
	<< extension >> (p. 795) Represents the initial number of active topic queries a DDS_DataWriter (p. 485) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_active_topic_queries (p. 1680)
	<< extension >> (p. 795) Represents the maximum number of active topic queries a DDS_DataWriter (p. 485) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	writer_loaned_sample_allocation (p. 1681)
	<< extension >> (p. 795) Represents the allocation settings of loaned samples managed by a DDS_DataWriter (p. 485).
DDS_SystemResourceLimitsQosPolicy (p. 2024)	max_objects_per_thread (p. 2025)
	[DEPRECATED] (p. 965) << extension >> (p. 795) The maximum number of objects that can be stored per thread for a DDS_DomainParticipantFactory (p. 32).

DDS_SystemResourceLimitsQosPolicy (p. 2024)	initial_objects_per_thread (p. 2025)
	<< <i>extension</i> >> (p. 795) The number of objects per thread for a DDS_DomainParticipantFactory (p. 32) for which infrastructure will initially be allocated.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_writer_allocation (p. 1735)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to local DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_reader_allocation (p. 1735)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to local DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_publisher_allocation (p. 1736)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to local Publisher.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_subscriber_allocation (p. 1736)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to local Subscriber.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_topic_allocation (p. 1736)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to local Topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_writer_allocation (p. 1737)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to remote DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_reader_allocation (p. 1737)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to remote DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_participant_allocation (p. 1737)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to remote DomainParticipants.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	matching_writer_reader_pair_allocation (p. 1738)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to matching local writer and remote/local reader pairs.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	matching_reader_writer_pair_allocation (p. 1738)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to matching local reader and remote/local writer pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	ignored_entity_allocation (p. 1738)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to ignored entities.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	content_filtered_topic_allocation (p. 1739)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to content filtered topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	content_filter_allocation (p. 1739)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to content filter.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	read_condition_allocation (p. 1739)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to read condition pool.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	query_condition_allocation (p. 1740)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to query condition pool.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	outstanding_asynchronous_sample_allocation (p. 1740)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to the maximum number of samples (from all DDS_DataWriter (p. 485)) waiting to be asynchronously written.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	flow_controller_allocation (p. 1740)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to flow controllers.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_writer_hash_buckets (p. 1741)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Hash_Buckets settings applied to local DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_reader_hash_buckets (p. 1741)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for local DataReaders.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_publisher_hash_buckets (p. 1741)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for local Publisher.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_subscriber_hash_buckets (p. 1742)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for local Subscriber.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_topic_hash_buckets (p. 1742)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for local Topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_writer_hash_buckets (p. 1742)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for remote DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_reader_hash_buckets (p. 1743)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for remote DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_participant_hash_buckets (p. 1743)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for remote DomainParticipants.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	matching_writer_reader_pair_hash_buckets (p. 1743)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for matching local writer and remote/local reader pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	matching_reader_writer_pair_hash_buckets (p. 1744)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for matching local reader and remote/local writer pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	ignored_entity_hash_buckets (p. 1744)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for ignored entities.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	content_filtered_topic_hash_buckets (p. 1744)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for content filtered topics.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	content_filter_hash_buckets (p. 1745)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for content filters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	flow_controller_hash_buckets (p. 1745)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for flow controllers.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	max_gather_destinations (p. 1745)
	<< <i>extension</i> >> (p. 795) Maximum number of destinations per RTI Connext send.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	participant_user_data_max_length (p. 1746)
	<< <i>extension</i> >> (p. 795) Maximum length of user data in DDS_DomainParticipantQos (p. 1726) and DDS_ParticipantBuiltinTopicData (p. 1870)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	topic_data_max_length (p. 1746)
	<< <i>extension</i> >> (p. 795) Maximum length of topic data in DDS_TopicQos (p. 2042), DDS_TopicBuiltinTopicData (p. 2034), DDS_PublicationBuiltinTopicData (p. 1905) and DDS_SubscriptionBuiltinTopicData (p. 2012)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	publisher_group_data_max_length (p. 1746)
	<< <i>extension</i> >> (p. 795) Maximum length of group data in DDS_PublisherQos (p. 1918) and DDS_PublicationBuiltinTopicData (p. 1905)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	subscriber_group_data_max_length (p. 1747)
	<< <i>extension</i> >> (p. 795) Maximum length of group data in DDS_SubscriberQos (p. 2010) and DDS_SubscriptionBuiltinTopicData (p. 2012)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	writer_user_data_max_length (p. 1747)
	<< <i>extension</i> >> (p. 795) Maximum length of user data in DDS_DataWriterQos (p. 1666) and DDS_PublicationBuiltinTopicData (p. 1905)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	reader_user_data_max_length (p. 1747)

	<< <i>extension</i> >> (p. 795) Maximum length of user data in DDS_DataReaderQos (p. 1611) and DDS_↔SubscriptionBuiltinTopicData (p. 2012)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	max_partitions (p. 1748)
	<< <i>extension</i> >> (p. 795) Maximum number of partition name strings allowable in a DDS_PartitionQosPolicy (p. 1881).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	max_partition_cumulative_characters (p. 1748)
	<< <i>extension</i> >> (p. 795) Maximum number of combined characters allowable in all partition names in a DDS_PartitionQosPolicy (p. 1881).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	type_code_max_serialized_length (p. 1749)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Maximum size of serialized string for type code.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	type_object_max_serialized_length (p. 1749)
	<< <i>extension</i> >> (p. 795) The maximum length, in bytes, that the buffer to serialize a TypeObject can consume.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	type_object_max_deserialized_length (p. 1750)
	<< <i>extension</i> >> (p. 795) The maximum number of bytes that a deserialized TypeObject can consume.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	deserialized_type_object_dynamic_allocation_↔threshold (p. 1751)
	<< <i>extension</i> >> (p. 795) A threshold, in bytes, for dynamic memory allocation for the deserialized Type↔Object.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	serialized_type_object_dynamic_allocation_↔threshold (p. 1750)
	<< <i>extension</i> >> (p. 795) A threshold, in bytes, for dynamic memory allocation for the serialized TypeObject.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	contentfilter_property_max_length (p. 1751)
	<< <i>extension</i> >> (p. 795) This field is the maximum length of all data related to a Content-filtered topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	channel_seq_max_length (p. 1752)

	<< <i>extension</i> >> (p. 795) Maximum number of channels that can be specified in DDS_MultiChannelQosPolicy (p. 1856) for MultiChannel DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	channel_filter_expression_max_length (p. 1752)
	<< <i>extension</i> >> (p. 795) Maximum length of a channel DDS_ChannelSettings_t::filter_expression (p. 1564) in a MultiChannel DataWriter.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	participant_property_list_max_length (p. 1752)
	<< <i>extension</i> >> (p. 795) Maximum number of properties associated with the DDS_DomainParticipant (p. 77).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	participant_property_string_max_length (p. 1753)
	<< <i>extension</i> >> (p. 795) Maximum string length of the properties associated with the DDS_DomainParticipant (p. 77).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	writer_property_list_max_length (p. 1753)
	<< <i>extension</i> >> (p. 795) Maximum number of properties associated with a DDS_DataWriter (p. 485).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	writer_property_string_max_length (p. 1753)
	<< <i>extension</i> >> (p. 795) Maximum string length of the properties associated with a DDS_DataWriter (p. 485).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	reader_property_list_max_length (p. 1754)
	<< <i>extension</i> >> (p. 795) Maximum number of properties associated with a DDS_DataReader (p. 615).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	reader_property_string_max_length (p. 1754)
	<< <i>extension</i> >> (p. 795) Maximum string length of the properties associated with a DDS_DataReader (p. 615).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	max_endpoint_groups (p. 1754)
	<< <i>extension</i> >> (p. 795) Maximum number of DDS_EndpointGroup_t (p. 1784) allowable in a DDS_AvailabilityQosPolicy (p. 1547).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	max_endpoint_group_cumulative_characters (p. 1755)

	<< <i>extension</i> >> (p. 795) Maximum number of combined <code>role_name</code> characters allowed in all DDS_EndpointGroup_t (p. 1784) in a DDS_AvailabilityQosPolicy (p. 1547).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	transport_info_list_max_length (p. 1755)
	<< <i>extension</i> >> (p. 795) Maximum number of installed transports to send and receive information about in DDS_ParticipantBuiltinTopicData::transport_info (p. 1873).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	ignored_entity_replacement_kind (p. 1755)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Replacement policy for the ignored entities. It sets what entity can be replaced when resource limits set in DDS_DomainParticipantResourceLimitsQosPolicy::ignored_entity_allocation (p. 1738) are reached.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_topic_query_allocation (p. 1756)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to remote TopicQueries.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_topic_query_hash_buckets (p. 1756)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for remote TopicQueries.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	writer_data_tag_list_max_length (p. 1757)
	<< <i>extension</i> >> (p. 795) Maximum number of data tags associated with a DDS_DataWriter (p. 485).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	writer_data_tag_string_max_length (p. 1757)
	<< <i>extension</i> >> (p. 795) Maximum string length of the data tags associated with a DDS_DataWriter (p. 485).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	reader_data_tag_list_max_length (p. 1757)
	<< <i>extension</i> >> (p. 795) Maximum number of data tags associated with a DDS_DataReader (p. 615).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	reader_data_tag_string_max_length (p. 1758)
	<< <i>extension</i> >> (p. 795) Maximum string length of the data tags associated with a DDS_DataReader (p. 615).

DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	shmem_ref_transfer_mode_max_segments (p. 1758)
	<< <i>extension</i> >> (p. 795) Maximum number of segments created by all DataWriters belonging to a DDS_DomainParticipant (p. 77).
DDS_EventQosPolicy (p. 1793)	initial_count (p. 1794)
	<< <i>extension</i> >> (p. 795) The initial number of events.
DDS_EventQosPolicy (p. 1793)	max_count (p. 1794)
	<< <i>extension</i> >> (p. 795) The maximum number of events.
DDS_DatabaseQosPolicy (p. 1581)	initial_records (p. 1584)
	<< <i>extension</i> >> (p. 795) The initial number of total records.
DDS_DatabaseQosPolicy (p. 1581)	max_skiplist_level (p. 1584)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The maximum level of the skiplist.
DDS_DatabaseQosPolicy (p. 1581)	initial_weak_references (p. 1585)
	<< <i>extension</i> >> (p. 795) The initial number of weak references.
DDS_DatabaseQosPolicy (p. 1581)	max_weak_references (p. 1584)
	<< <i>extension</i> >> (p. 795) The maximum number of weak references.
DDS_ReceiverPoolQosPolicy (p. 1934)	buffer_size (p. 1935)
	<< <i>extension</i> >> (p. 795) The receive buffer size in bytes.
DDS_ReceiverPoolQosPolicy (p. 1934)	buffer_alignment (p. 1936)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The receive buffer alignment.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_reader_resource_limits (p. 1696)
	<< <i>extension</i> >> (p. 795) Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 1689)	publication_reader_resource_limits (p. 1697)
	<< <i>extension</i> >> (p. 795) Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 1689)	subscription_reader_resource_limits (p. 1698)
	<< <i>extension</i> >> (p. 795) Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_configuration_reader_resource_limits (p. 1714)
	<< <i>extension</i> >> (p. 795) Resource limits for the built-in topic participant configuration reader.

DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	initial_samples (p. 1558)
	<< <i>extension</i> >> (p. 795) Initial number of samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_samples (p. 1558)
	<< <i>extension</i> >> (p. 795) Maximum number of samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	initial_infos (p. 1559)
	<< <i>extension</i> >> (p. 795) Initial number of sample infos.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_infos (p. 1559)
	<< <i>extension</i> >> (p. 795) Maximum number of sample infos.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	initial_outstanding_reads (p. 1559)
	<< <i>extension</i> >> (p. 795) The initial number of outstanding reads that have not called finish yet on the same built-in topic DDS_DataReader (p. 615).
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_outstanding_reads (p. 1560)
	<< <i>extension</i> >> (p. 795) The maximum number of outstanding reads that have not called finish yet on the same built-in topic DDS_DataReader (p. 615).
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_samples_per_read (p. 1560)
	<< <i>extension</i> >> (p. 795) Maximum number of samples that can be read/taken on a same built-in topic DDS_DataReader (p. 615).
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	disable_fragmentation_support (p. 1560)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Determines whether the built-in topic DDS_DataReader (p. 615) can receive fragmented samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_fragmented_samples (p. 1561)
	<< <i>extension</i> >> (p. 795) The maximum number of samples for which the built-in topic DDS_DataReader (p. 615) may store fragments at a given point in time.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	initial_fragmented_samples (p. 1561)
	<< <i>extension</i> >> (p. 795) The initial number of samples for which a built-in topic DDS_DataReader (p. 615) may store fragments.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_fragmented_samples_per_remote_writer (p. 1562)

	<< extension >> (p. 795) The maximum number of samples per remote writer for which a built-in topic DDS_DataReader (p. 615) may store fragments.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_fragments_per_sample (p. 1562)
	<< extension >> (p. 795) Maximum number of fragments for a single sample.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	dynamically_allocate_fragmented_samples (p. 1563)
	<< extension >> (p. 795) Determines whether the built-in topic DDS_DataReader (p. 615) pre-allocates storage for storing fragmented samples.
DDS_ThreadSettings_t (p. 2029)	stack_size (p. 2030)
	The thread stack-size.
DDS_LoggingQosPolicy (p. 1831)	max_bytes_per_file (p. 1833)
	<< extension >> (p. 795) Specifies the maximum number of bytes a single file can contain
DDS_LoggingQosPolicy (p. 1831)	max_files (p. 1834)
	<< extension >> (p. 795) Specifies the maximum number of files to create before overwriting the previous ones.

5.68 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. 1898)	string_profile (p. 1899)
	<< basic >> (p. 783) << extension >> (p. 795) Sequence of strings containing a XML document to load.
DDS_ProfileQosPolicy (p. 1898)	url_profile (p. 1899)
	<< basic >> (p. 783) << extension >> (p. 795) Sequence of URL groups (p. 973) containing a set of XML documents to load.
DDS_ProfileQosPolicy (p. 1898)	ignore_user_profile (p. 1899)
	<< basic >> (p. 783) << extension >> (p. 795) Ignores the file USER_QOS_PROFILES.xml in the current working directory.

DDS_ProfileQosPolicy (p. 1898)	ignore_environment_profile (p. 1899)
	<< basic >> (p. 783) << extension >> (p. 795) Ignores the value of the NDDS_QOS_PROFILES (p. 973).
DDS_ProfileQosPolicy (p. 1898)	ignore_resource_profile (p. 1900)
	<< basic >> (p. 783) << extension >> (p. 795) Ignores the file NDDS_QOS_PROFILES.xml .
DDS_MonitoringDedicatedParticipantSettings (p. 1836)	enable (p. 1836)
	<< basic >> (p. 783) << extension >> (p. 795) Enables the use of a dedicated DDS_DomainParticipant (p. 77) to distribute the RTI Connex application telemetry data.

Advanced Settings

DDS_MonitoringEventDistributionSettings (p. 1840)	datawriter_qos_profile_name (p. 1842)
	<< extension >> (p. 795) The fully qualified name of the profile used to configure the DDS_DataWriter (p. 485) that distributes event metrics.
DDS_MonitoringPeriodicDistributionSettings (p. 1852)	datawriter_qos_profile_name (p. 1852)
	<< extension >> (p. 795) The fully qualified name of the profile used to configure the DDS_DataWriter (p. 485) that distributes periodic metrics.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	datawriter_qos_profile_name (p. 1845)
	<< extension >> (p. 795) The fully qualified name of the profile used to configure the DDS_DataWriter (p. 485) that distributes log messages.
DDS_MonitoringDedicatedParticipantSettings (p. 1836)	participant_qos_profile_name (p. 1837)
	<< extension >> (p. 795) The fully qualified name of the profile used to configure the DDS_DomainParticipant (p. 77) that will be used to distribute telemetry data.
DDS_MonitoringDistributionSettings (p. 1838)	dedicated_participant (p. 1839)
	<< extension >> (p. 795) Configures the use of a dedicated DDS_DomainParticipant (p. 77) to distribute the RTI Connex application telemetry data.
DDS_MonitoringDistributionSettings (p. 1838)	publisher_qos_profile_name (p. 1839)

	<< extension >> (p. 795) The fully qualified name of the profile used to configure the Publishers that distribute telemetry data.

5.69 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. 1936)	kind (p. 1939)
	<< basic >> (p. 783) << spec >> (p. 790) Kind of reliability.
DDS_ReliabilityQosPolicy (p. 1936)	max_blocking_time (p. 1939)
	<< basic >> (p. 783) << spec >> (p. 790) The maximum time a DataWriter can block on a write() call.
DDS_HistoryQosPolicy (p. 1805)	kind (p. 1807)
	<< basic >> (p. 783) << spec >> (p. 790) Specifies the kind of history to be kept.
DDS_HistoryQosPolicy (p. 1805)	depth (p. 1807)
	<< basic >> (p. 783) << spec >> (p. 790) Specifies the number of samples per instance to be kept, when the <code>kind</code> is DDS_↵KEEP_LAST_HISTORY_QOS (p. 1293)
DDS_RtpsReliableWriterProtocol_t (p. 1960)	heartbeat_period (p. 1963)
	<< basic >> (p. 783) << extension >> (p. 795) The period at which to send heartbeats.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	fast_heartbeat_period (p. 1963)
	<< basic >> (p. 783) << extension >> (p. 795) An alternative heartbeat period used when a reliable writer needs to flush its un-acknowledged samples more quickly.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	late_joiner_heartbeat_period (p. 1964)

	<< basic >> (p. 783) << extension >> (p. 795) 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. 1960)	min_send_window_size (p. 1973)
	<< basic >> (p. 783) << extension >> (p. 795) Minimum size of send window of unacknowledged samples
DDS_RtpsReliableWriterProtocol_t (p. 1960)	max_send_window_size (p. 1973)
	<< basic >> (p. 783) << extension >> (p. 795) Maximum size of send window of unacknowledged samples

Advanced Settings

DDS_AvailabilityQosPolicy (p. 1547)	enable_required_subscriptions (p. 1550)
	<< extension >> (p. 795) Enables support for required subscriptions in a DDS_DataWriter (p. 485)
DDS_AvailabilityQosPolicy (p. 1547)	max_data_availability_waiting_time (p. 1550)
	<< extension >> (p. 795) Defines how much time to wait before delivering a sample to the application without having received some of the previous samples.
DDS_AvailabilityQosPolicy (p. 1547)	max_endpoint_availability_waiting_time (p. 1550)
	<< extension >> (p. 795) Defines how much time to wait to discover DataWriters providing samples for the same data source (virtual GUID).
DDS_AvailabilityQosPolicy (p. 1547)	required_matched_endpoint_groups (p. 1551)
	<< extension >> (p. 795) A sequence of endpoint groups.
DDS_ReliabilityQosPolicy (p. 1936)	acknowledgment_kind (p. 1939)
	<< extension >> (p. 795) Kind of reliable acknowledgment
DDS_ReliabilityQosPolicy (p. 1936)	instance_state_consistency_kind (p. 1940)
	<< extension >> (p. 795) Whether instance state consistency is enabled
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_app_ack_response_length (p. 1633)
	<< extension >> (p. 795) Maximum length of application-level acknowledgment response data
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_app_ack_remote_readers (p. 1680)
	<< extension >> (p. 795) The maximum number of application-level acknowledging remote readers supported by a DDS_DataWriter (p. 485)

DDS_DataReaderProtocolQosPolicy (p. 1596)	rtps_reliable_reader (p. 1599)
	<< <i>extension</i> >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a DDS_DataReader (p. 615). This parameter only has effect if the reader is configured with DDS_RELIABLE_RELIABILITY_QOS (p. 1323) DDS_↵_ReliabilityQosPolicyKind (p. 1322).
DDS_DataReaderProtocolQosPolicy (p. 1596)	disable_positive_acks (p. 1598)
	<< <i>extension</i> >> (p. 795) Whether the reader sends positive acknowledgements to writers.
DDS_DataWriterProtocolQosPolicy (p. 1649)	push_on_write (p. 1650)
	<< <i>extension</i> >> (p. 795) Whether to push sample out when write is called.
DDS_DataWriterProtocolQosPolicy (p. 1649)	disable_positive_acks (p. 1651)
	<< <i>extension</i> >> (p. 795) Controls whether or not the writer expects positive acknowledgements from matching readers.
DDS_DataWriterProtocolQosPolicy (p. 1649)	propagate_app_ack_with_no_response (p. 1653)
	<< <i>extension</i> >> (p. 795) Controls whether or not a DDS_DataWriter (p. 485) receives DDS_DataWriter↵Listener::on_application_acknowledgment (p. 1648) notifications with an empty or invalid response.
DDS_DataWriterProtocolQosPolicy (p. 1649)	rtps_reliable_writer (p. 1653)
	<< <i>extension</i> >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a DDS_DataWriter (p. 485). This parameter only has effect if both the writer and the matching reader are configured with DDS_RELIABLE_RELIABILITY_QOS (p. 1323) DDS_ReliabilityQosPolicyKind (p. 1322).
DDS_DiscoveryConfigQosPolicy (p. 1689)	publication_reader (p. 1696)
	<< <i>extension</i> >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in publication reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	subscription_reader (p. 1697)
	<< <i>extension</i> >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in subscription reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_message_writer (p. 1702)

	<< extension >> (p. 795) 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. 1323) DDS_ReliabilityQosPolicyKind (p. 1322).
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_message_reader_reliability_kind (p. 1701)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Reliability policy for a built-in participant message reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_message_reader (p. 1702)
	<< extension >> (p. 795) 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. 1701) is set to DDS_RELIABLE_RELIABILITY_QOS (p. 1323).
DDS_DiscoveryConfigQosPolicy (p. 1689)	publication_writer (p. 1698)
	<< extension >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	subscription_writer (p. 1700)
	<< extension >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in subscription writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	service_request_writer (p. 1705)
	<< extension >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in DDS_ServiceRequest (p. 2001) writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	service_request_reader (p. 1707)
	<< extension >> (p. 795) RTPS reliable reader protocol-related configuration settings for a built-in DDS_ServiceRequest (p. 2001) reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	secure_volatile_writer (p. 1709)
	<< extension >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in secure volatile writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	secure_volatile_reader (p. 1710)
	<< extension >> (p. 795) RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.

DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_configuration_reader (p. 1714)
	<< <i>extension</i> >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in participant configuration reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_configuration_writer (p. 1712)
	<< <i>extension</i> >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in participant configuration writer.
DDS_RtpsReliableReaderProtocol_t (p. 1954)	min_heartbeat_response_delay (p. 1955)
	<< <i>extension</i> >> (p. 795) The minimum delay to respond to a heartbeat.
DDS_RtpsReliableReaderProtocol_t (p. 1954)	max_heartbeat_response_delay (p. 1955)
	<< <i>extension</i> >> (p. 795) The maximum delay to respond to a heartbeat.
DDS_RtpsReliableReaderProtocol_t (p. 1954)	heartbeat_suppression_duration (p. 1956)
	<< <i>extension</i> >> (p. 795) The duration a reader ignores consecutively received heartbeats
DDS_RtpsReliableReaderProtocol_t (p. 1954)	nack_period (p. 1956)
	<< <i>extension</i> >> (p. 795) The period at which to send NACKs
DDS_RtpsReliableReaderProtocol_t (p. 1954)	receive_window_size (p. 1957)
	<< <i>extension</i> >> (p. 795) The number of received out-of-order samples a reader can keep at a time
DDS_RtpsReliableReaderProtocol_t (p. 1954)	round_trip_time (p. 1957)
	<< <i>extension</i> >> (p. 795) The duration from sending a NACK to receiving a repair of a sample
DDS_RtpsReliableReaderProtocol_t (p. 1954)	app_ack_period (p. 1958)
	<< <i>extension</i> >> (p. 795) The period at which application-level acknowledgment messages are sent
DDS_RtpsReliableReaderProtocol_t (p. 1954)	min_app_ack_response_keep_duration (p. 1958)
	<< <i>extension</i> >> (p. 795) Minimum duration for which application-level acknowledgment response data is kept
DDS_RtpsReliableReaderProtocol_t (p. 1954)	samples_per_app_ack (p. 1959)
	<< <i>extension</i> >> (p. 795) The minimum number of samples acknowledged by one application-level acknowledgment message
DDS_RtpsReliableWriterProtocol_t (p. 1960)	low_watermark (p. 1961)

	<< <i>extension</i> >> (p. 795) 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. 1234) is considered to have changed.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	high_watermark (p. 1962)
	<< <i>extension</i> >> (p. 795) 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. 1234) is considered to have changed.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	virtual_heartbeat_period (p. 1965)
	<< <i>extension</i> >> (p. 795) 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. 1960)	samples_per_virtual_heartbeat (p. 1965)
	<< <i>extension</i> >> (p. 795) The number of samples that a reliable writer has to publish before sending a virtual heartbeat.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	max_heartbeat_retries (p. 1966)
	<< <i>extension</i> >> (p. 795) The maximum number of <i>periodic</i> heartbeat retries before marking a remote reader as inactive.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	inactivate_nonprogressing_readers (p. 1966)
	<< <i>extension</i> >> (p. 795) Whether to treat remote readers as inactive when their NACKs do not progress
DDS_RtpsReliableWriterProtocol_t (p. 1960)	heartbeats_per_max_samples (p. 1967)
	<< <i>extension</i> >> (p. 795) The number of piggyback heartbeats sent per max send window.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	min_nack_response_delay (p. 1968)
	<< <i>extension</i> >> (p. 795) The minimum delay to respond to a NACK or NACK_FRAG.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	max_nack_response_delay (p. 1968)
	<< <i>extension</i> >> (p. 795) The maximum delay to respond to a NACK or NACK_FRAG.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	nack_suppression_duration (p. 1969)
	<< <i>extension</i> >> (p. 795) The duration for ignoring consecutive NACKs/NACK_FRAGs that may trigger redundant repairs.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	max_bytes_per_nack_response (p. 1969)

	<< <i>extension</i> >> (p. 795) The maximum total message size when resending rejected samples or data fragments.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_enable_adaptive_sample_↔keep_duration (p. 1971)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Enables dynamic adjustment of sample keep duration in response to congestion.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_min_sample_keep_duration (p. 1970)
	<< <i>extension</i> >> (p. 795) The minimum duration a sample is queued for ACK-disabled readers.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_max_sample_keep_duration (p. 1971)
	<< <i>extension</i> >> (p. 795) The maximum duration a sample is queued for ACK-disabled readers.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_decrease_sample_keep_↔duration_factor (p. 1972)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Controls rate of contraction of dynamic sample keep duration.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_increase_sample_keep_↔duration_factor (p. 1972)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Controls rate of growth of dynamic sample keep duration.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	send_window_update_period (p. 1974)
	<< <i>extension</i> >> (p. 795) Period in which send window may be dynamically changed
DDS_RtpsReliableWriterProtocol_t (p. 1960)	send_window_increase_factor (p. 1975)
	<< <i>extension</i> >> (p. 795) Increases send window size by this percentage when reacting dynamically to network conditions
DDS_RtpsReliableWriterProtocol_t (p. 1960)	send_window_decrease_factor (p. 1976)
	<< <i>extension</i> >> (p. 795) Decreases send window size by this percentage when reacting dynamically to network conditions
DDS_RtpsReliableWriterProtocol_t (p. 1960)	enable_multicast_periodic_heartbeat (p. 1977)
	<< <i>extension</i> >> (p. 795) Whether periodic heartbeat messages are sent over multicast
DDS_RtpsReliableWriterProtocol_t (p. 1960)	multicast_resend_threshold (p. 1977)

	<< extension >> (p. 795) The minimum number of requesting readers needed to trigger a multicast resend
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_repair_piggyback_heartbeat (p. 1978)
	<< extension >> (p. 795) Prevents piggyback heartbeats from being sent with repair samples

5.70 Sample Memory Management QoS

Use these QoS fields to configure how Connex manages memory related to the samples that are stored and managed by the DataReader and DataWriter queues.

Use these QoS fields to configure how Connex manages memory related to the samples that are stored and managed by the DataReader and DataWriter queues.

Basic Settings

DDS_ResourceLimitsQosPolicy (p. 1949)	max_samples (p. 1951)
	<< basic >> (p. 783) << spec >> (p. 790) Represents the maximum samples the middleware can store for any one DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)).
DDS_ResourceLimitsQosPolicy (p. 1949)	max_instances (p. 1951)
	<< basic >> (p. 783) << spec >> (p. 790) Represents the maximum number of instances a DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)) can manage.
DDS_ResourceLimitsQosPolicy (p. 1949)	initial_samples (p. 1952)
	<< basic >> (p. 783) << extension >> (p. 795) Represents the initial samples the middleware will store for any one DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)).
DDS_ResourceLimitsQosPolicy (p. 1949)	initial_instances (p. 1952)
	<< basic >> (p. 783) << extension >> (p. 795) Represents the initial number of instances a DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)) will manage.

Advanced Settings

DDS_PersistentStorageSettings (p. 1883)	writer_instance_cache_allocation (p. 1887)
	<< extension >> (p. 795) Configures the resource limits associated with the instance durable writer history cache.

DDS_PersistentStorageSettings (p. 1883)	writer_sample_cache_allocation (p. 1888)
	<< <i>extension</i> >> (p. 795) Configures the resource limits associated with the sample durable writer history cache.
DDS_DurabilityServiceQosPolicy (p. 1764)	history_kind (p. 1765)
	<< <i>spec</i> >> (p. 790) The kind of history to apply in recouping durable data.
DDS_DurabilityServiceQosPolicy (p. 1764)	history_depth (p. 1765)
	<< <i>spec</i> >> (p. 790) Setting to use for the DDS_↔DurabilityQosPolicy::writer_depth (p. 1762) when recouping durable data.
DDS_DurabilityServiceQosPolicy (p. 1764)	max_samples (p. 1766)
	<< <i>spec</i> >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1764)	max_instances (p. 1766)
	<< <i>spec</i> >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1764)	max_samples_per_instance (p. 1766)
	<< <i>spec</i> >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_ResourceLimitsQosPolicy (p. 1949)	max_samples_per_instance (p. 1952)
	<< <i>spec</i> >> (p. 790) Represents the maximum number of samples of any one instance a DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)) can manage.
DDS_WriterDataLifecycleQosPolicy (p. 2106)	autodispose_unregistered_instances (p. 2107)
	<< <i>spec</i> >> (p. 790) Boolean flag that controls whether or not a writer will also dispose of an instance when the DDS_DataWriter (p. 485) unregisters from the instance by means of the unregister operations.
DDS_WriterDataLifecycleQosPolicy (p. 2106)	autopurge_unregistered_instances_delay (p. 2107)
	<< <i>extension</i> >> (p. 795) Maximum duration for which the DDS_DataWriter (p. 485) will maintain information regarding an instance once it has unregistered the instance.
DDS_WriterDataLifecycleQosPolicy (p. 2106)	autopurge_disposed_instances_delay (p. 2108)
	<< <i>extension</i> >> (p. 795) Maximum duration for which the DDS_DataWriter (p. 485) will maintain information regarding an instance once it has disposed the instance.
DDS_ReaderDataLifecycleQosPolicy (p. 1930)	autopurge_nowriter_samples_delay (p. 1932)

	<< <i>spec</i> >> (p. 790) Minimum duration for which the DDS_DataReader (p.615) will maintain samples for an instance once its <code>instance_state</code> becomes DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE (p. 715).
DDS_ReaderDataLifecycleQosPolicy (p. 1930)	autopurge_disposed_samples_delay (p. 1932)
	<< <i>spec</i> >> (p. 790) Minimum duration for which the DDS_DataReader (p.615) will maintain samples for an instance once its <code>instance_state</code> becomes DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE (p. 715).
DDS_ReaderDataLifecycleQosPolicy (p. 1930)	autopurge_disposed_instances_delay (p. 1932)
	<< <i>extension</i> >> (p. 795) Minimum duration for which the DDS_DataReader (p.615) will maintain information about a received instance once its <code>instance_state</code> becomes DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE (p. 715) and there are no samples for the instance in the DataReader queue.
DDS_ReaderDataLifecycleQosPolicy (p. 1930)	autopurge_nowriter_instances_delay (p. 1933)
	<< <i>extension</i> >> (p. 795) Minimum duration for which the DDS_DataReader (p.615) will maintain information about a received instance once its <code>instance_state</code> becomes DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE (p. 715) and there are no samples for the instance in the DataReader queue.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_fragmented_samples (p. 1626)
	<< <i>extension</i> >> (p. 795) The maximum number of samples for which the DDS_DataReader (p.615) may store fragments at a given point in time.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_fragmented_samples (p. 1627)
	<< <i>extension</i> >> (p. 795) The initial number of samples for which a DDS_DataReader (p.615) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_fragmented_samples_per_remote_writer (p. 1627)
	<< <i>extension</i> >> (p. 795) The maximum number of samples per remote writer for which a DDS_DataReader (p. 615) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_fragments_per_sample (p. 1628)
	<< <i>extension</i> >> (p. 795) Maximum number of fragments for a single sample.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	dynamically_allocate_fragmented_samples (p. 1628)

	<< extension >> (p. 795) Determines whether the DDS_DataReader (p. 615) pre-allocates storage for storing fragmented samples.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	initial_batches (p. 1676)
	<< extension >> (p. 795) Represents the initial number of batches a DDS_DataWriter (p. 485) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_batches (p. 1677)
	<< extension >> (p. 795) Represents the maximum number of batches a DDS_DataWriter (p. 485) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	writer_loaned_sample_allocation (p. 1681)
	<< extension >> (p. 795) Represents the allocation settings of loaned samples managed by a DDS_DataWriter (p. 485).
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	initialize_writer_loaned_sample (p. 1681)
	<< extension >> (p. 795) Whether or not to initialize loaned samples returned by a DDS_DataWriter (p. 485)
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_reader_resource_limits (p. 1696)
	<< extension >> (p. 795) Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 1689)	publication_reader_resource_limits (p. 1697)
	<< extension >> (p. 795) Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 1689)	subscription_reader_resource_limits (p. 1698)
	<< extension >> (p. 795) Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_configuration_reader_resource_limits (p. 1714)
	<< extension >> (p. 795) Resource limits for the built-in topic participant configuration reader.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	initial_samples (p. 1558)
	<< extension >> (p. 795) Initial number of samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_samples (p. 1558)
	<< extension >> (p. 795) Maximum number of samples.

5.71 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. 1686)	kind (p. 1688)
	<< basic >> (p. 783) << spec >> (p. 790) Specifies the desired kind of destination order.

Advanced Settings

DDS_PresentationQosPolicy (p. 1890)	access_scope (p. 1893)
	<< spec >> (p. 790) Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
DDS_PresentationQosPolicy (p. 1890)	coherent_access (p. 1893)
	<< spec >> (p. 790) Specifies support for <i>coherent</i> access. Controls whether coherent access is supported within the scope <code>access_scope</code> .
DDS_PresentationQosPolicy (p. 1890)	ordered_access (p. 1893)
	<< spec >> (p. 790) Specifies support for <i>ordered</i> access to the samples received at the subscription end. Controls whether ordered access is supported within the scope <code>access_scope</code> .
DDS_PresentationQosPolicy (p. 1890)	drop_incomplete_coherent_set (p. 1894)
	<< extension >> (p. 795) Indicates whether or not a DDS_DataReader (p. 615) 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 <code>SAMPLE_LOST</code> Status.
DDS_DestinationOrderQosPolicy (p. 1686)	scope (p. 1688)
	<< spec >> (p. 790) Specifies the desired scope of the source destination order.
DDS_DestinationOrderQosPolicy (p. 1686)	source_timestamp_tolerance (p. 1688)
	<< extension >> (p. 795) Allowed tolerance between source timestamps of consecutive samples.

5.72 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. 1847)	security_event_forwarding_level (p. 1847)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) Log messages with NDDS_CONFIG_LOG_FACILITY_↔SECURITY_EVENT (p. 1447) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.

Advanced Settings

DDS_TypeSupportQosPolicy (p. 2081)	cdr_padding_kind (p. 2082)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Determines whether or not the padding bytes will be set to zero during CDR serialization.
DDS_WireProtocolQosPolicy (p. 2093)	compute_crc (p. 2100)
	<< <i>extension</i> >> (p. 795) Adds RTPS CRC submessage to every message when this field is set to DDS_BOOLEAN_TRUE (p. 1201).
DDS_WireProtocolQosPolicy (p. 2093)	check_crc (p. 2100)
	<< <i>extension</i> >> (p. 795) 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. 1201).
DDS_DiscoveryConfigQosPolicy (p. 1689)	secure_volatile_writer (p. 1709)
	<< <i>extension</i> >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in secure volatile writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	secure_volatile_writer_publish_mode (p. 1710)
	<< <i>extension</i> >> (p. 795) Publish mode policy for the built-in secure volatile writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	secure_volatile_reader (p. 1710)
	<< <i>extension</i> >> (p. 795) RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.
DDS_DataTags (p. 1641)	tags (p. 1641)
	<< <i>spec</i> >> (p. 790) Sequence of data tags.

5.73 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. 1552)	enable (p. 1552)
	<< basic >> (p. 783) << extension >> (p. 795) Specifies whether or not batching is enabled.
DDS_BatchQosPolicy (p. 1552)	max_data_bytes (p. 1553)
	<< basic >> (p. 783) << extension >> (p. 795) The maximum cumulative length of all serialized samples in a batch.
DDS_BatchQosPolicy (p. 1552)	max_samples (p. 1553)
	<< basic >> (p. 783) << extension >> (p. 795) The maximum number of samples in a batch.

Advanced Settings

DDS_BatchQosPolicy (p. 1552)	max_flush_delay (p. 1554)
	<< extension >> (p. 795) The maximum flush delay.
DDS_BatchQosPolicy (p. 1552)	source_timestamp_resolution (p. 1554)
	<< extension >> (p. 795) Batch source timestamp resolution.
DDS_BatchQosPolicy (p. 1552)	thread_safe_write (p. 1555)
	<< extension >> (p. 795) Determines whether or not the write operation is thread safe.

5.74 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. 2051)	mask (p. 2052)
	<< basic >> (p. 783) << extension >> (p. 795) Specifies the built-in transports that are registered automatically when the DDS_DomainParticipant (p. 77) is enabled.
DDS_TransportMulticastQosPolicy (p. 2058)	value (p. 2059)
	<< basic >> (p. 783) << extension >> (p. 795) A sequence of multicast communications settings.
DDS_DiscoveryQosPolicy (p. 1715)	initial_peers (p. 1716)
	<< basic >> (p. 783) << extension >> (p. 795) 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. 1715)	multicast_receive_addresses (p. 1717)
	<< basic >> (p. 783) << extension >> (p. 795) Specifies the multicast group addresses on which discovery-related meta-traffic can be received by the DomainParticipant.
DDS_DiscoveryQosPolicy (p. 1715)	accept_unknown_peers (p. 1718)
	<< basic >> (p. 783) << extension >> (p. 795) Whether to accept a new participant that is not in the initial peers list.

Advanced Settings

DDS_TransportPriorityQosPolicy (p. 2062)	value (p. 2064)
	<< spec >> (p. 790) 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. 2064)	enabled_transports (p. 2065)
	<< extension >> (p. 795) A sequence of transport aliases that specifies the transport instances available for use by the entity.
DDS_TransportUnicastQosPolicy (p. 2065)	value (p. 2067)
	<< extension >> (p. 795) A sequence of unicast communication settings.
DDS_TransportMulticastMappingQosPolicy (p. 2056)	value (p. 2058)
	<< extension >> (p. 795) A sequence of multicast communication mappings.
DDS_TransportMulticastQosPolicy (p. 2058)	kind (p. 2059)
	<< extension >> (p. 795) A value that specifies a way to determine how to obtain the multicast address.

DDS_DiscoveryQosPolicy (p. 1715)	enabled_transports (p. 1716)
	<< <i>extension</i> >> (p. 795) The transports available for use by the Discovery mechanism.
DDS_DiscoveryQosPolicy (p. 1715)	metatraffic_transport_priority (p. 1717)
	<< <i>extension</i> >> (p. 795) The transport priority to use for the Discovery meta-traffic.
DDS_DiscoveryQosPolicy (p. 1715)	enable_endpoint_discovery (p. 1718)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Whether to automatically enable endpoint discovery for all the remote participants.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	transport_info_list_max_length (p. 1755)
	<< <i>extension</i> >> (p. 795) Maximum number of installed transports to send and receive information about in DDS_ParticipantBuiltinTopicData::transport_info (p. 1873).
DDS_DiscoveryConfigQosPolicy (p. 1689)	locator_reachability_assert_period (p. 1707)
	<< <i>extension</i> >> (p. 795) Period at which this DomainParticipant will assert the locators discovered from other DomainParticipants.
DDS_DiscoveryConfigQosPolicy (p. 1689)	locator_reachability_change_detection_period (p. 1708)
	<< <i>extension</i> >> (p. 795) Period at which this DomainParticipant will check if its locators are reachable from other DomainParticipants.
DDS_DiscoveryConfigQosPolicy (p. 1689)	dns_tracker_polling_period (p. 1711)
	<< <i>extension</i> >> (p. 795) 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. 1960)	enable_multicast_periodic_heartbeat (p. 1977)
	<< <i>extension</i> >> (p. 795) Whether periodic heartbeat messages are sent over multicast
DDS_RtpsReliableWriterProtocol_t (p. 1960)	multicast_resend_threshold (p. 1977)
	<< <i>extension</i> >> (p. 795) The minimum number of requesting readers needed to trigger a multicast resend
DDS_ChannelSettings_t (p. 1564)	multicast_settings (p. 1564)
	<< <i>extension</i> >> (p. 795) A sequence of DDS_↔TransportMulticastSettings_t (p. 2060) used to configure the multicast addresses associated with a channel.

DDS_ChannelSettings_t (p. 1564)	priority (p. 1565)
	<< extension >> (p. 795) Publication priority

5.75 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. 2047)	publication_period (p. 2048)
	<< basic >> (p. 783) << extension >> (p. 795) Sets the periodic interval at which samples are published.

Advanced Settings

DDS_AsynchronousPublisherQosPolicy (p. 1539)	disable_topic_query_publication (p. 1542)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Disable topic query publication.
DDS_AsynchronousPublisherQosPolicy (p. 1539)	topic_query_publication_thread (p. 1542)
	<< extension >> (p. 795) Settings of the DDS_TopicQuery (p. 707) publication thread.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_topic_queries (p. 1634)
	<< extension >> (p. 795) The initial number of Topic↔Queries allocated by a DDS_DataReader (p. 615).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_topic_queries (p. 1634)
	<< extension >> (p. 795) The maximum number of active TopicQueries that a DDS_DataReader (p. 615) can create.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	initial_active_topic_queries (p. 1680)
	<< extension >> (p. 795) Represents the initial number of active topic queries a DDS_DataWriter (p. 485) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_active_topic_queries (p. 1680)

	<< extension >> (p. 795) Represents the maximum number of active topic queries a DDS_DataWriter (p. 485) will manage.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_topic_query_allocation (p. 1756)
	<< extension >> (p. 795) Allocation settings applied to remote TopicQueries.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_topic_query_hash_buckets (p. 1756)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Number of hash buckets for remote TopicQueries.
DDS_TopicQueryDispatchQosPolicy (p. 2047)	enable (p. 2048)
	<< extension >> (p. 795) Allows this writer to dispatch TopicQueries
DDS_TopicQueryDispatchQosPolicy (p. 2047)	samples_per_period (p. 2049)
	<< extension >> (p. 795) Sets the maximum number of samples to publish in each publication_period.

5.76 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. 1638)	value (p. 1640)
	<< spec >> (p. 790) Sequence of representation identifiers.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	kind (p. 2077)
	<< spec >> (p. 790) Type consistency kind.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	ignore_sequence_bounds (p. 2077)
	<< spec >> (p. 790) Controls whether sequence bounds are taken into consideration for type assignability.

DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	ignore_string_bounds (p. 2078)
	<< <i>spec</i> >> (p. 790) Controls whether string bounds are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	ignore_member_names (p. 2078)
	<< <i>spec</i> >> (p. 790) Controls whether member names are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	prevent_type_widening (p. 2078)
	<< <i>spec</i> >> (p. 790) Controls whether type widening is allowed.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	force_type_validation (p. 2079)
	<< <i>spec</i> >> (p. 790) Controls whether type information must be available in order to complete matching between a DDS_DataWriter (p. 485) and a DDS_DataReader (p. 615).
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	ignore_enum_literal_names (p. 2079)
	<< <i>spec</i> >> (p. 790) Controls whether enumeration constant names are taken into consideration for type assignability.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	type_code_max_serialized_length (p. 1749)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Maximum size of serialized string for type code.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	type_object_max_serialized_length (p. 1749)
	<< <i>extension</i> >> (p. 795) The maximum length, in bytes, that the buffer to serialize a TypeObject can consume.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	type_object_max_deserialized_length (p. 1750)
	<< <i>extension</i> >> (p. 795) The maximum number of bytes that a deserialized TypeObject can consume.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	deserialized_type_object_dynamic_allocation_threshold (p. 1751)
	<< <i>extension</i> >> (p. 795) A threshold, in bytes, for dynamic memory allocation for the deserialized TypeObject.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	serialized_type_object_dynamic_allocation_↔ threshold (p. 1750)
	<< <i>extension</i> >> (p. 795) A threshold, in bytes, for dynamic memory allocation for the serialized TypeObject.
DDS_DiscoveryConfigQosPolicy (p. 1689)	endpoint_type_object_lb_serialization_threshold (p. 1711)
	<< <i>extension</i> >> (p. 795) Option to reduce the size required to propagate a TypeObject in Simple Endpoint Discovery

5.77 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.

5.77.1 Detailed Description

Some QoS are mutable after an entity is enabled, and others are not.

5.78 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. 949)).
- 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. 1245)

Basic Settings

DDS_DurabilityQosPolicy (p. 1759)	kind (p. 1762)
	<< basic >> (p. 783) << spec >> (p. 790) The kind of durability.
DDS_DurabilityQosPolicy (p. 1759)	writer_depth (p. 1762)
	<< basic >> (p. 783) << extension >> (p. 795) Indicates the number of samples per instance that a durable DDS_DataWriter (p. 485) will send to a late-joining DDS_DataReader (p. 615).
DDS_DataRepresentationQosPolicy (p. 1638)	compression_settings (p. 1640)
	<< basic >> (p. 783) << extension >> (p. 795) Structure that contains the compression settings.
DDS_OwnershipQosPolicy (p. 1863)	kind (p. 1868)
	<< basic >> (p. 783) << spec >> (p. 790) The kind of ownership
DDS_LivelinessQosPolicy (p. 1823)	kind (p. 1825)
	<< basic >> (p. 783) << spec >> (p. 790) The kind of liveliness desired.
DDS_LivelinessQosPolicy (p. 1823)	lease_duration (p. 1825)
	<< basic >> (p. 783) << spec >> (p. 790) The duration within which a DDS_DataWriter (p. 485) must be asserted, or else it is assumed to be not alive.
DDS_LivelinessQosPolicy (p. 1823)	assertions_per_lease_duration (p. 1826)
	<< basic >> (p. 783) << extension >> (p. 795) The number of assertions a DDS_DataWriter (p. 485) will send during a its DDS_LivelinessQosPolicy::lease_duration (p. 1825).
DDS_ReliabilityQosPolicy (p. 1936)	kind (p. 1939)
	<< basic >> (p. 783) << spec >> (p. 790) Kind of reliability.
DDS_ReliabilityQosPolicy (p. 1936)	max_blocking_time (p. 1939)
	<< basic >> (p. 783) << spec >> (p. 790) The maximum time a DataWriter can block on a write() call.
DDS_DestinationOrderQosPolicy (p. 1686)	kind (p. 1688)
	<< basic >> (p. 783) << spec >> (p. 790) Specifies the desired kind of destination order.
DDS_HistoryQosPolicy (p. 1805)	kind (p. 1807)
	<< basic >> (p. 783) << spec >> (p. 790) Specifies the kind of history to be kept.
DDS_HistoryQosPolicy (p. 1805)	depth (p. 1807)

	<< basic >> (p. 783) << spec >> (p. 790) Specifies the number of samples per instance to be kept, when the kind is DDS_KEEP_LAST_HISTORY_QOS (p. 1293)
DDS_ResourceLimitsQosPolicy (p. 1949)	max_samples (p. 1951)
	<< basic >> (p. 783) << spec >> (p. 790) Represents the maximum samples the middleware can store for any one DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)).
DDS_ResourceLimitsQosPolicy (p. 1949)	max_instances (p. 1951)
	<< basic >> (p. 783) << spec >> (p. 790) Represents the maximum number of instances a DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)) can manage.
DDS_ResourceLimitsQosPolicy (p. 1949)	initial_samples (p. 1952)
	<< basic >> (p. 783) << extension >> (p. 795) Represents the initial samples the middleware will store for any one DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)).
DDS_ResourceLimitsQosPolicy (p. 1949)	initial_instances (p. 1952)
	<< basic >> (p. 783) << extension >> (p. 795) Represents the initial number of instances a DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)) will manage.
DDS_PublishModeQosPolicy (p. 1920)	kind (p. 1922)
	<< basic >> (p. 783) << extension >> (p. 795) Publishing mode
DDS_WireProtocolQosPolicy (p. 2093)	participant_id (p. 2097)
	<< basic >> (p. 783) << extension >> (p. 795) A value used to distinguish among different participants belonging to the same domain on the same host.
DDS_TransportBuiltinQosPolicy (p. 2051)	mask (p. 2052)
	<< basic >> (p. 783) << extension >> (p. 795) Specifies the built-in transports that are registered automatically when the DDS_DomainParticipant (p. 77) is enabled.
DDS_TransportMulticastQosPolicy (p. 2058)	value (p. 2059)
	<< basic >> (p. 783) << extension >> (p. 795) A sequence of multicast communications settings.
DDS_DiscoveryQosPolicy (p. 1715)	initial_peers (p. 1716)
	<< basic >> (p. 783) << extension >> (p. 795) 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. 1715)	multicast_receive_addresses (p. 1717)

	<< basic >> (p. 783) << extension >> (p. 795) Specifies the multicast group addresses on which discovery-related meta-traffic can be received by the DomainParticipant.
DDS_DiscoveryQosPolicy (p. 1715)	accept_unknown_peers (p. 1718)
	<< basic >> (p. 783) << extension >> (p. 795) Whether to accept a new participant that is not in the initial peers list.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_liveliness_lease_duration (p. 1692)
	<< basic >> (p. 783) << extension >> (p. 795) The liveliness lease duration for the participant.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_liveliness_assert_period (p. 1692)
	<< basic >> (p. 783) << extension >> (p. 795) The period to assert liveliness for the participant.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_announcement_period (p. 1693)
	<< basic >> (p. 783) << extension >> (p. 795) The period at which a participant announces itself to potential peers when using the Simple Participant Discovery Protocol 2.0 (SPDP2).
DDS_DiscoveryConfigQosPolicy (p. 1689)	initial_participant_announcements (p. 1694)
	<< basic >> (p. 783) << extension >> (p. 795) The number of initial announcements sent when a participant is first enabled.
DDS_DiscoveryConfigQosPolicy (p. 1689)	new_remote_participant_announcements (p. 1695)
	<< basic >> (p. 783) << extension >> (p. 795) The number of participant announcements sent when a remote participant is newly discovered.
DDS_DiscoveryConfigQosPolicy (p. 1689)	min_initial_participant_announcement_period (p. 1695)
	<< basic >> (p. 783) << extension >> (p. 795) The minimum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
DDS_DiscoveryConfigQosPolicy (p. 1689)	max_initial_participant_announcement_period (p. 1695)
	<< basic >> (p. 783) << extension >> (p. 795) The maximum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	heartbeat_period (p. 1963)
	<< basic >> (p. 783) << extension >> (p. 795) The period at which to send heartbeats.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	fast_heartbeat_period (p. 1963)
	<< basic >> (p. 783) << extension >> (p. 795) An alternative heartbeat period used when a reliable writer needs to flush its unacknowledged samples more quickly.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	late_joiner_heartbeat_period (p. 1964)

	<< basic >> (p. 783) << extension >> (p. 795) 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. 1960)	min_send_window_size (p. 1973)
	<< basic >> (p. 783) << extension >> (p. 795) Minimum size of send window of unacknowledged samples
DDS_RtpsReliableWriterProtocol_t (p. 1960)	max_send_window_size (p. 1973)
	<< basic >> (p. 783) << extension >> (p. 795) Maximum size of send window of unacknowledged samples
DDS_EntityNameQosPolicy (p. 1790)	name (p. 1790)
	<< basic >> (p. 783) << extension >> (p. 795) The name of the entity.
DDS_BatchQosPolicy (p. 1552)	enable (p. 1552)
	<< basic >> (p. 783) << extension >> (p. 795) Specifies whether or not batching is enabled.
DDS_BatchQosPolicy (p. 1552)	max_data_bytes (p. 1553)
	<< basic >> (p. 783) << extension >> (p. 795) The maximum cumulative length of all serialized samples in a batch.
DDS_BatchQosPolicy (p. 1552)	max_samples (p. 1553)
	<< basic >> (p. 783) << extension >> (p. 795) The maximum number of samples in a batch.
DDS_ProfileQosPolicy (p. 1898)	string_profile (p. 1899)
	<< basic >> (p. 783) << extension >> (p. 795) Sequence of strings containing a XML document to load.
DDS_ProfileQosPolicy (p. 1898)	url_profile (p. 1899)
	<< basic >> (p. 783) << extension >> (p. 795) Sequence of URL groups (p. 973) containing a set of XML documents to load.
DDS_ProfileQosPolicy (p. 1898)	ignore_user_profile (p. 1899)
	<< basic >> (p. 783) << extension >> (p. 795) Ignores the file USER_QOS_PROFILES.xml in the current working directory.
DDS_ProfileQosPolicy (p. 1898)	ignore_environment_profile (p. 1899)
	<< basic >> (p. 783) << extension >> (p. 795) Ignores the value of the NDDS_QOS_PROFILES (p. 973).
DDS_ProfileQosPolicy (p. 1898)	ignore_resource_profile (p. 1900)
	<< basic >> (p. 783) << extension >> (p. 795) Ignores the file NDDS_QOS_PROFILES.xml .
DDS_TopicQueryDispatchQosPolicy (p. 2047)	publication_period (p. 2048)

	<< basic >> (p. 783) << extension >> (p. 795) Sets the periodic interval at which samples are published.

Advanced Settings

DDS_AvailabilityQosPolicy (p. 1547)	enable_required_subscriptions (p. 1550)
	<< extension >> (p. 795) Enables support for required subscriptions in a DDS_DataWriter (p. 485)
DDS_DurabilityQosPolicy (p. 1759)	direct_communication (p. 1762)
	<< extension >> (p. 795) Indicates whether or not a TRANSIENT or PERSISTENT DDS_DataReader (p. 615) should receive samples directly from a TRANSIENT or PERSISTENT DDS_DataWriter (p. 485)
DDS_DurabilityQosPolicy (p. 1759)	storage_settings (p. 1763)
	<< extension >> (p. 795) Configures durable writer history and durable reader state.
DDS_PersistentStorageSettings (p. 1883)	enable (p. 1885)
	<< extension >> (p. 795) Enables durable writer history in a DDS_DataWriter (p. 485) and durable reader state in a DDS_DataReader (p. 615).
DDS_PersistentStorageSettings (p. 1883)	file_name (p. 1885)
	<< extension >> (p. 795) The file name where the durable writer history or durable reader state will be stored.
DDS_PersistentStorageSettings (p. 1883)	trace_file_name (p. 1885)
	<< extension >> (p. 795) 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. 1883)	journal_kind (p. 1886)
	<< extension >> (p. 795) Sets the journal mode of the persistent storage.
DDS_PersistentStorageSettings (p. 1883)	synchronization_kind (p. 1886)
	<< extension >> (p. 795) Sets the level of synchronization with the physical disk.
DDS_PersistentStorageSettings (p. 1883)	vacuum (p. 1886)
	<< extension >> (p. 795) Sets the auto-vacuum status of the storage.
DDS_PersistentStorageSettings (p. 1883)	restore (p. 1887)

	<< extension >> (p. 795) Indicates if the persisted writer history or reader state must be restored.
DDS_PersistentStorageSettings (p. 1883)	writer_instance_cache_allocation (p. 1887)
	<< extension >> (p. 795) Configures the resource limits associated with the instance durable writer history cache.
DDS_PersistentStorageSettings (p. 1883)	writer_sample_cache_allocation (p. 1888)
	<< extension >> (p. 795) Configures the resource limits associated with the sample durable writer history cache.
DDS_PersistentStorageSettings (p. 1883)	writer_memory_state (p. 1889)
	<< extension >> (p. 795) 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. 1883)	reader_checkpoint_frequency (p. 1889)
	<< extension >> (p. 795) Controls how often the reader state is stored into the database.
DDS_DurabilityServiceQosPolicy (p. 1764)	service_cleanup_delay (p. 1765)
	<< spec >> (p. 790) Controls when the service is able to remove all information regarding a data instances.
DDS_DurabilityServiceQosPolicy (p. 1764)	history_kind (p. 1765)
	<< spec >> (p. 790) The kind of history to apply in re-couping durable data.
DDS_DurabilityServiceQosPolicy (p. 1764)	history_depth (p. 1765)
	<< spec >> (p. 790) Setting to use for the DDS_↵ DurabilityQosPolicy::writer_depth (p. 1762) when re-couping durable data.
DDS_DurabilityServiceQosPolicy (p. 1764)	max_samples (p. 1766)
	<< spec >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1764)	max_instances (p. 1766)
	<< spec >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1764)	max_samples_per_instance (p. 1766)
	<< spec >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_CompressionSettings_t (p. 1568)	compression_ids (p. 1568)
	<< extension >> (p. 795) Mask that represents the compression algorithms enabled.

DDS_CompressionSettings_t (p. 1568)	writer_compression_level (p. 1569)
	<< <i>extension</i> >> (p. 795) The level of compression to use when compressing data.
DDS_CompressionSettings_t (p. 1568)	writer_compression_threshold (p. 1569)
	<< <i>extension</i> >> (p. 795) The threshold, in bytes, above which a serialized sample will be eligible to be compressed.
DDS_DataRepresentationQosPolicy (p. 1638)	value (p. 1640)
	<< <i>spec</i> >> (p. 790) Sequence of representation identifiers.
DDS_PresentationQosPolicy (p. 1890)	access_scope (p. 1893)
	<< <i>spec</i> >> (p. 790) Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
DDS_PresentationQosPolicy (p. 1890)	coherent_access (p. 1893)
	<< <i>spec</i> >> (p. 790) Specifies support for <i>coherent</i> access. Controls whether coherent access is supported within the scope <code>access_scope</code> .
DDS_PresentationQosPolicy (p. 1890)	ordered_access (p. 1893)
	<< <i>spec</i> >> (p. 790) Specifies support for <i>ordered</i> access to the samples received at the subscription end. Controls whether ordered access is supported within the scope <code>access_scope</code> .
DDS_PresentationQosPolicy (p. 1890)	drop_incomplete_coherent_set (p. 1894)
	<< <i>extension</i> >> (p. 795) Indicates whether or not a DDS_DataReader (p. 615) 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 <code>SAMPLE_LOST</code> Status.
DDS_ReliabilityQosPolicy (p. 1936)	acknowledgment_kind (p. 1939)
	<< <i>extension</i> >> (p. 795) Kind of reliable acknowledgment
DDS_ReliabilityQosPolicy (p. 1936)	instance_state_consistency_kind (p. 1940)
	<< <i>extension</i> >> (p. 795) Whether instance state consistency is enabled
DDS_DestinationOrderQosPolicy (p. 1686)	scope (p. 1688)
	<< <i>spec</i> >> (p. 790) Specifies the desired scope of the source destination order.
DDS_DestinationOrderQosPolicy (p. 1686)	source_timestamp_tolerance (p. 1688)

	<< extension >> (p. 795) Allowed tolerance between source timestamps of consecutive samples.
DDS_ResourceLimitsQosPolicy (p. 1949)	max_samples_per_instance (p. 1952)
	<< spec >> (p. 790) Represents the maximum number of samples of any one instance a DDS_DataWriter (p. 485) (or DDS_DataReader (p. 615)) can manage.
DDS_ResourceLimitsQosPolicy (p. 1949)	instance_hash_buckets (p. 1953)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Number of hash buckets for instances.
DDS_TransportPriorityQosPolicy (p. 2062)	value (p. 2064)
	<< spec >> (p. 790) 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. 2081)	plugin_data (p. 2082)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Value to pass into the type plugin's de-/serialization function
DDS_TypeSupportQosPolicy (p. 2081)	cdr_padding_kind (p. 2082)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Determines whether or not the padding bytes will be set to zero during CDR serialization.
DDS_PublishModeQosPolicy (p. 1920)	flow_controller_name (p. 1922)
	<< extension >> (p. 795) Name of the associated flow controller
DDS_PublishModeQosPolicy (p. 1920)	priority (p. 1923)
	<< extension >> (p. 795) Publication priority
DDS_AsynchronousPublisherQosPolicy (p. 1539)	disable_asynchronous_write (p. 1540)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Disable asynchronous publishing.
DDS_AsynchronousPublisherQosPolicy (p. 1539)	thread (p. 1540)
	<< extension >> (p. 795) Settings of the publishing thread.
DDS_AsynchronousPublisherQosPolicy (p. 1539)	disable_asynchronous_batch (p. 1541)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Disable asynchronous batch flushing.
DDS_AsynchronousPublisherQosPolicy (p. 1539)	asynchronous_batch_thread (p. 1541)
	<< extension >> (p. 795) Settings of the batch flushing thread.
DDS_AsynchronousPublisherQosPolicy (p. 1539)	disable_topic_query_publication (p. 1542)

	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Disable topic query publication.
DDS_AsymynchronousPublisherQosPolicy (p. 1539)	topic_query_publication_thread (p. 1542)
	<< <i>extension</i> >> (p. 795) Settings of the DDS_Topic ↔ Query (p. 707) publication thread.
DDS_WriterDataLifecycleQosPolicy (p. 2106)	autodispose_unregistered_instances (p. 2107)
	<< <i>spec</i> >> (p. 790) Boolean flag that controls whether or not a writer will also dispose of an instance when the DDS_DataWriter (p. 485) unregisters from the instance by means of the unregister operations.
DDS_WriterDataLifecycleQosPolicy (p. 2106)	autopurge_unregistered_instances_delay (p. 2107)
	<< <i>extension</i> >> (p. 795) Maximum duration for which the DDS_DataWriter (p. 485) will maintain information regarding an instance once it has unregistered the in- stance.
DDS_WriterDataLifecycleQosPolicy (p. 2106)	autopurge_disposed_instances_delay (p. 2108)
	<< <i>extension</i> >> (p. 795) Maximum duration for which the DDS_DataWriter (p. 485) will maintain information regarding an instance once it has disposed the instance.
DDS_ReaderDataLifecycleQosPolicy (p. 1930)	autopurge_nowriter_samples_delay (p. 1932)
	<< <i>spec</i> >> (p. 790) Minimum duration for which the DDS_DataReader (p. 615) will maintain samples for an instance once its <code>instance_state</code> becomes DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE (p. 715).
DDS_ReaderDataLifecycleQosPolicy (p. 1930)	autopurge_disposed_samples_delay (p. 1932)
	<< <i>spec</i> >> (p. 790) Minimum duration for which the DDS_DataReader (p. 615) will maintain samples for an instance once its <code>instance_state</code> be- comes DDS_NOT_ALIVE_DISPOSED_INSTANCE_ ↔ STATE (p. 715).
DDS_ReaderDataLifecycleQosPolicy (p. 1930)	autopurge_disposed_instances_delay (p. 1932)
	<< <i>extension</i> >> (p. 795) Minimum duration for which the DDS_DataReader (p. 615) will maintain information about a received instance once its <code>instance_state</code> becomes DDS_NOT_ALIVE_DISPOSED_INSTANCE_ ↔ _STATE (p. 715) and there are no samples for the in- stance in the DataReader queue.
DDS_ReaderDataLifecycleQosPolicy (p. 1930)	autopurge_nowriter_instances_delay (p. 1933)

	<< extension >> (p. 795) Minimum duration for which the DDS_DataReader (p. 615) will maintain information about a received instance once its <code>instance_state</code> becomes DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE (p. 715) and there are no samples for the instance in the DataReader queue.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	kind (p. 2077)
	<< spec >> (p. 790) Type consistency kind.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	ignore_sequence_bounds (p. 2077)
	<< spec >> (p. 790) Controls whether sequence bounds are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	ignore_string_bounds (p. 2078)
	<< spec >> (p. 790) Controls whether string bounds are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	ignore_member_names (p. 2078)
	<< spec >> (p. 790) Controls whether member names are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	prevent_type_widening (p. 2078)
	<< spec >> (p. 790) Controls whether type widening is allowed.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	force_type_validation (p. 2079)
	<< spec >> (p. 790) Controls whether type information must be available in order to complete matching between a DDS_DataWriter (p. 485) and a DDS_DataReader (p. 615).
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	ignore_enum_literal_names (p. 2079)
	<< spec >> (p. 790) Controls whether enumeration constant names are taken into consideration for type assignability.
DDS_WireProtocolQosPolicy (p. 2093)	rtps_host_id (p. 2097)
	<< extension >> (p. 795) The RTPS Host ID of the domain participant.
DDS_WireProtocolQosPolicy (p. 2093)	rtps_app_id (p. 2098)

	<< extension >> (p. 795) The RTPS App ID of the domain participant.
DDS_WireProtocolQosPolicy (p. 2093)	rtps_instance_id (p. 2098)
	<< extension >> (p. 795) The RTPS Instance ID of the DDS_DomainParticipant (p. 77).
DDS_WireProtocolQosPolicy (p. 2093)	rtps_well_known_ports (p. 2099)
	<< extension >> (p. 795) Configures the RTPS well-known port mappings.
DDS_WireProtocolQosPolicy (p. 2093)	rtps_reserved_port_mask (p. 2099)
	<< extension >> (p. 795) Specifies which well-known ports to reserve when enabling the participant.
DDS_WireProtocolQosPolicy (p. 2093)	rtps_auto_id_kind (p. 2100)
	<< extension >> (p. 795) Kind of auto mechanism used to calculate the GUID prefix.
DDS_WireProtocolQosPolicy (p. 2093)	compute_crc (p. 2100)
	<< extension >> (p. 795) Adds RTPS CRC submessage to every message when this field is set to DDS_↵ BOOLEAN_TRUE (p. 1201).
DDS_WireProtocolQosPolicy (p. 2093)	check_crc (p. 2100)
	<< extension >> (p. 795) 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. 1201).
DDS_TransportSelectionQosPolicy (p. 2064)	enabled_transports (p. 2065)
	<< extension >> (p. 795) A sequence of transport aliases that specifies the transport instances available for use by the entity.
DDS_TransportUnicastQosPolicy (p. 2065)	value (p. 2067)
	<< extension >> (p. 795) A sequence of unicast communication settings.
DDS_TransportMulticastMappingQosPolicy (p. 2056)	value (p. 2058)
	<< extension >> (p. 795) A sequence of multicast communication mappings.
DDS_TransportMulticastQosPolicy (p. 2058)	kind (p. 2059)
	<< extension >> (p. 795) A value that specifies a way to determine how to obtain the multicast address.
DDS_DiscoveryQosPolicy (p. 1715)	enabled_transports (p. 1716)
	<< extension >> (p. 795) The transports available for use by the Discovery mechanism.

DDS_DiscoveryQosPolicy (p. 1715)	metatraffic_transport_priority (p. 1717)
	<< <i>extension</i> >> (p. 795) The transport priority to use for the Discovery meta-traffic.
DDS_DiscoveryQosPolicy (p. 1715)	enable_endpoint_discovery (p. 1718)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Whether to automatically enable endpoint discovery for all the remote participants.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_remote_writers (p. 1621)
	<< <i>extension</i> >> (p. 795) The maximum number of remote writers from which a DDS_DataReader (p. 615) may read, including all instances.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_remote_writers_per_instance (p. 1622)
	<< <i>extension</i> >> (p. 795) The maximum number of remote writers from which a DDS_DataReader (p. 615) may read a single instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_samples_per_remote_writer (p. 1622)
	<< <i>extension</i> >> (p. 795) The maximum number of out-of-order samples from a given remote DDS_DataWriter (p. 485) that a DDS_DataReader (p. 615) may store when maintaining a reliable connection to the DDS_↔DataWriter (p. 485).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_infos (p. 1623)
	<< <i>extension</i> >> (p. 795) The maximum number of info units that a DDS_DataReader (p. 615) can use to store DDS_SampleInfo (p. 1985).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_remote_writers (p. 1623)
	<< <i>extension</i> >> (p. 795) The initial number of remote writers from which a DDS_DataReader (p. 615) may read, including all instances.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_remote_writers_per_instance (p. 1624)
	<< <i>extension</i> >> (p. 795) The initial number of remote writers from which a DDS_DataReader (p. 615) may read a single instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_infos (p. 1624)
	<< <i>extension</i> >> (p. 795) The initial number of info units that a DDS_DataReader (p. 615) can have, which are used to store DDS_SampleInfo (p. 1985).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_outstanding_reads (p. 1625)

	<< extension >> (p. 795) The initial number of outstanding calls to read/take (or one of their variants) on the same DDS_DataReader (p. 615) for which memory has not been returned by calling FooDataReader_return_loan (p. 646).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_outstanding_reads (p. 1625)
	<< extension >> (p. 795) The maximum number of outstanding read/take calls (or one of their variants) on the same DDS_DataReader (p. 615) for which memory has not been returned by calling FooDataReader_return_loan (p. 646).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_samples_per_read (p. 1625)
	<< extension >> (p. 795) The maximum number of data samples that the application can receive from the middleware in a single call to FooDataReader_read (p. 625) or FooDataReader_take (p. 626). If more data exists in the middleware, the application will need to issue multiple read/take calls.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	disable_fragmentation_support (p. 1626)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Determines whether the DDS_DataReader (p. 615) can receive fragmented samples.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_fragmented_samples (p. 1626)
	<< extension >> (p. 795) The maximum number of samples for which the DDS_DataReader (p. 615) may store fragments at a given point in time.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_fragmented_samples (p. 1627)
	<< extension >> (p. 795) The initial number of samples for which a DDS_DataReader (p. 615) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_fragmented_samples_per_remote_writer (p. 1627)
	<< extension >> (p. 795) The maximum number of samples per remote writer for which a DDS_DataReader (p. 615) may store fragments.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_fragments_per_sample (p. 1628)
	<< extension >> (p. 795) Maximum number of fragments for a single sample.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	dynamically_allocate_fragmented_samples (p. 1628)
	<< extension >> (p. 795) Determines whether the DDS_DataReader (p. 615) pre-allocates storage for storing fragmented samples.

DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_total_instances (p. 1629)
	<< <i>extension</i> >> (p. 795) Maximum number of instances for which a DataReader will keep state.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_remote_virtual_writers (p. 1630)
	<< <i>extension</i> >> (p. 795) The maximum number of remote virtual writers from which a DDS_DataReader (p. 615) may read, including all instances.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_remote_virtual_writers (p. 1630)
	<< <i>extension</i> >> (p. 795) The initial number of remote virtual writers from which a DDS_DataReader (p. 615) may read, including all instances.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_remote_virtual_writers_per_instance (p. 1631)
	<< <i>extension</i> >> (p. 795) The maximum number of virtual remote writers that can be associated with an instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_remote_virtual_writers_per_instance (p. 1631)
	<< <i>extension</i> >> (p. 795) The initial number of virtual remote writers per instance.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_remote_writers_per_sample (p. 1632)
	<< <i>extension</i> >> (p. 795) The maximum number of remote writers allowed to write the same sample.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_query_condition_filters (p. 1632)
	<< <i>extension</i> >> (p. 795) The maximum number of query condition filters a reader is allowed.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_app_ack_response_length (p. 1633)
	<< <i>extension</i> >> (p. 795) Maximum length of application-level acknowledgment response data
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	keep_minimum_state_for_instances (p. 1633)
	<< <i>extension</i> >> (p. 795) Whether or not keep a minimum instance state for up to DDS_DataReaderResourceLimitsQosPolicy::max_total_instances (p. 1629).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	initial_topic_queries (p. 1634)
	<< <i>extension</i> >> (p. 795) The initial number of TopicQueries allocated by a DDS_DataReader (p. 615).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	max_topic_queries (p. 1634)
	<< <i>extension</i> >> (p. 795) The maximum number of active TopicQueries that a DDS_DataReader (p. 615) can create.

DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	shmем_ref_transfer_mode_attached_segment_allocation (p. 1635)
	<< <i>extension</i> >> (p. 795) Allocation resource for the shared memory segments attached by the DDS_DataReader (p. 615).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	instance_replacement (p. 1635)
	<< <i>extension</i> >> (p. 795) Sets the kind of instances allowed to be replaced for each instance state (DDS_InstanceStateKind (p. 714)) when a DataReader reaches DDS_ResourceLimitsQosPolicy::max_instances (p. 1951).
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	autopurge_remote_not_alive_writer_delay (p. 1636)
	<< <i>extension</i> >> (p. 795) Maximum duration for which the DDS_DataReader (p. 615) will maintain information regarding a DDS_DataWriter (p. 485) once the DDS_DataWriter (p. 485) has become not alive.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	initial_concurrent_blocking_threads (p. 1675)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The initial number of threads that are allowed to concurrently block on write call on the same DDS_DataWriter (p. 485).
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_concurrent_blocking_threads (p. 1675)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The maximum number of threads that are allowed to concurrently block on write call on the same DDS_DataWriter (p. 485).
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_remote_reader_filters (p. 1676)
	<< <i>extension</i> >> (p. 795) The maximum number of remote DataReaders for which the DDS_DataWriter (p. 485) will perform content-based filtering.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	initial_batches (p. 1676)
	<< <i>extension</i> >> (p. 795) Represents the initial number of batches a DDS_DataWriter (p. 485) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_batches (p. 1677)
	<< <i>extension</i> >> (p. 795) Represents the maximum number of batches a DDS_DataWriter (p. 485) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	instance_replacement (p. 1677)
	<< <i>extension</i> >> (p. 795) Sets the kinds of instances allowed to be replaced when instance resource limits are reached.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	replace_empty_instances (p. 1678)

	<< <i>extension</i> >> (p. 795) Whether or not to replace empty instances during instance replacement
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	autoregister_instances (p. 1678)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Whether or not to automatically register new instances
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_virtual_writers (p. 1679)
	<< <i>extension</i> >> (p. 795) The maximum number of virtual writers supported by a DDS_DataWriter (p. 485)
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	initial_virtual_writers (p. 1679)
	<< <i>extension</i> >> (p. 795) The initial number of virtual writers supported by a DDS_DataWriter (p. 485)
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_remote_readers (p. 1679)
	<< <i>extension</i> >> (p. 795) The maximum number of remote readers supported by a DDS_DataWriter (p. 485)
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_app_ack_remote_readers (p. 1680)
	<< <i>extension</i> >> (p. 795) The maximum number of application-level acknowledging remote readers supported by a DDS_DataWriter (p. 485)
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	initial_active_topic_queries (p. 1680)
	<< <i>extension</i> >> (p. 795) Represents the initial number of active topic queries a DDS_DataWriter (p. 485) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_active_topic_queries (p. 1680)
	<< <i>extension</i> >> (p. 795) Represents the maximum number of active topic queries a DDS_DataWriter (p. 485) will manage.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	writer_loaned_sample_allocation (p. 1681)
	<< <i>extension</i> >> (p. 795) Represents the allocation settings of loaned samples managed by a DDS_DataWriter (p. 485).
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	initialize_writer_loaned_sample (p. 1681)
	<< <i>extension</i> >> (p. 795) Whether or not to initialize loaned samples returned by a DDS_DataWriter (p. 485)
DDS_DataWriterTransferModeQosPolicy (p. 1683)	shmem_ref_settings (p. 1683)
	<< <i>extension</i> >> (p. 795) Settings related to transferring data using shared memory references.
DDS_DataReaderProtocolQosPolicy (p. 1596)	virtual_guid (p. 1597)

	<< <i>extension</i> >> (p. 795) The virtual GUID (Global Unique Identifier).
DDS_DataReaderProtocolQosPolicy (p. 1596)	rtps_object_id (p. 1597)
	<< <i>extension</i> >> (p. 795) The RTPS Object ID.
DDS_DataReaderProtocolQosPolicy (p. 1596)	rtps_reliable_reader (p. 1599)
	<< <i>extension</i> >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a DDS_DataReader (p. 615). This parameter only has effect if the reader is configured with DDS_RELIABLE_RELIABILITY_QOS (p. 1323) DDS_↵_ReliabilityQosPolicyKind (p. 1322).
DDS_DataReaderProtocolQosPolicy (p. 1596)	expects_inline_qos (p. 1597)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Specifies whether this DataReader expects inline QoS with every sample.
DDS_DataReaderProtocolQosPolicy (p. 1596)	disable_positive_acks (p. 1598)
	<< <i>extension</i> >> (p. 795) Whether the reader sends positive acknowledgements to writers.
DDS_DataReaderProtocolQosPolicy (p. 1596)	propagate_dispose_of_unregistered_instances (p. 1598)
	<< <i>extension</i> >> (p. 795) Indicates whether or not an instance can move to the DDS_NOT_ALIVE_↵DISPOSED_INSTANCE_STATE (p. 715) state without being in the DDS_ALIVE_INSTANCE_STATE (p. 715) state.
DDS_DataReaderProtocolQosPolicy (p. 1596)	propagate_unregister_of_disposed_instances (p. 1599)
	<< <i>extension</i> >> (p. 795) Indicates whether or not an instance can move to the DDS_NOT_ALIVE_NO_↵_WRITERS_INSTANCE_STATE (p. 715) state directly from the DDS_NOT_ALIVE_DISPOSED_INSTANCE_↵_STATE (p. 715).
DDS_DataWriterProtocolQosPolicy (p. 1649)	virtual_guid (p. 1650)
	<< <i>extension</i> >> (p. 795) The virtual GUID (Global Unique Identifier).
DDS_DataWriterProtocolQosPolicy (p. 1649)	rtps_object_id (p. 1650)
	<< <i>extension</i> >> (p. 795) The RTPS Object ID.
DDS_DataWriterProtocolQosPolicy (p. 1649)	push_on_write (p. 1650)
	<< <i>extension</i> >> (p. 795) Whether to push sample out when write is called.
DDS_DataWriterProtocolQosPolicy (p. 1649)	disable_positive_acks (p. 1651)

	<< <i>extension</i> >> (p. 795) Controls whether or not the writer expects positive acknowledgements from matching readers.
DDS_DataWriterProtocolQosPolicy (p. 1649)	disable_inline_keyhash (p. 1651)
	<< <i>extension</i> >> (p. 795) Controls whether or not a keyhash is propagated on the wire with each sample.
DDS_DataWriterProtocolQosPolicy (p. 1649)	serialize_key_with_dispose (p. 1652)
	<< <i>extension</i> >> (p. 795) Controls whether or not the serialized key is propagated on the wire with dispose samples.
DDS_DataWriterProtocolQosPolicy (p. 1649)	propagate_app_ack_with_no_response (p. 1653)
	<< <i>extension</i> >> (p. 795) Controls whether or not a DDS_DataWriter (p. 485) receives DDS_DataWriterListener::on_application_acknowledgment (p. 1648) notifications with an empty or invalid response.
DDS_DataWriterProtocolQosPolicy (p. 1649)	rtps_reliable_writer (p. 1653)
	<< <i>extension</i> >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a DDS_DataWriter (p. 485). This parameter only has effect if both the writer and the matching reader are configured with DDS_RELIABLE_RELIABILITY_QOS (p. 1323) DDS_ReliabilityQosPolicyKind (p. 1322).
DDS_DataWriterProtocolQosPolicy (p. 1649)	initial_virtual_sequence_number (p. 1653)
	<< <i>extension</i> >> (p. 795) Determines, the initial virtual sequence number for this DataWriter.
DDS_SystemResourceLimitsQosPolicy (p. 2024)	max_objects_per_thread (p. 2025)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The maximum number of objects that can be stored per thread for a DDS_DomainParticipantFactory (p. 32).
DDS_SystemResourceLimitsQosPolicy (p. 2024)	initial_objects_per_thread (p. 2025)
	<< <i>extension</i> >> (p. 795) The number of objects per thread for a DDS_DomainParticipantFactory (p. 32) for which infrastructure will initially be allocated.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_writer_allocation (p. 1735)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to local DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_reader_allocation (p. 1735)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to local DataReaders.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_publisher_allocation (p. 1736)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to local Publisher.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_subscriber_allocation (p. 1736)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to local Subscriber.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_topic_allocation (p. 1736)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to local Topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_writer_allocation (p. 1737)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to remote DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_reader_allocation (p. 1737)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to remote DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_participant_allocation (p. 1737)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to remote DomainParticipants.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	matching_writer_reader_pair_allocation (p. 1738)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to matching local writer and remote/local reader pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	matching_reader_writer_pair_allocation (p. 1738)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to matching local reader and remote/local writer pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	ignored_entity_allocation (p. 1738)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to ignored entities.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	content_filtered_topic_allocation (p. 1739)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to content filtered topic.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	content_filter_allocation (p. 1739)
	<< extension >> (p. 795) Allocation settings applied to content filter.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	read_condition_allocation (p. 1739)
	<< extension >> (p. 795) Allocation settings applied to read condition pool.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	query_condition_allocation (p. 1740)
	<< extension >> (p. 795) Allocation settings applied to query condition pool.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	outstanding_asynchronous_sample_allocation (p. 1740)
	<< extension >> (p. 795) Allocation settings applied to the maximum number of samples (from all DDS_DataWriter (p. 485)) waiting to be asynchronously written.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	flow_controller_allocation (p. 1740)
	<< extension >> (p. 795) Allocation settings applied to flow controllers.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_writer_hash_buckets (p. 1741)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Hash_Buckets settings applied to local DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_reader_hash_buckets (p. 1741)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Number of hash buckets for local DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_publisher_hash_buckets (p. 1741)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Number of hash buckets for local Publisher.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_subscriber_hash_buckets (p. 1742)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Number of hash buckets for local Subscriber.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_topic_hash_buckets (p. 1742)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Number of hash buckets for local Topic.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_writer_hash_buckets (p. 1742)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for remote DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_reader_hash_buckets (p. 1743)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for remote DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_participant_hash_buckets (p. 1743)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for remote DomainParticipants.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	matching_writer_reader_pair_hash_buckets (p. 1743)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for matching local writer and remote/local reader pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	matching_reader_writer_pair_hash_buckets (p. 1744)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for matching local reader and remote/local writer pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	ignored_entity_hash_buckets (p. 1744)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for ignored entities.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	content_filtered_topic_hash_buckets (p. 1744)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for content filtered topics.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	content_filter_hash_buckets (p. 1745)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for content filters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	flow_controller_hash_buckets (p. 1745)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for flow controllers.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	max_gather_destinations (p. 1745)
	<< <i>extension</i> >> (p. 795) Maximum number of destinations per RTI Connex send.

DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	participant_user_data_max_length (p. 1746)
	<< <i>extension</i> >> (p. 795) Maximum length of user data in DDS_DomainParticipantQos (p. 1726) and DDS_ParticipantBuiltinTopicData (p. 1870)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	topic_data_max_length (p. 1746)
	<< <i>extension</i> >> (p. 795) Maximum length of topic data in DDS_TopicQos (p. 2042), DDS_TopicBuiltinTopicData (p. 2034), DDS_PublicationBuiltinTopicData (p. 1905) and DDS_SubscriptionBuiltinTopicData (p. 2012)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	publisher_group_data_max_length (p. 1746)
	<< <i>extension</i> >> (p. 795) Maximum length of group data in DDS_PublisherQos (p. 1918) and DDS_PublicationBuiltinTopicData (p. 1905)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	subscriber_group_data_max_length (p. 1747)
	<< <i>extension</i> >> (p. 795) Maximum length of group data in DDS_SubscriberQos (p. 2010) and DDS_SubscriptionBuiltinTopicData (p. 2012)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	writer_user_data_max_length (p. 1747)
	<< <i>extension</i> >> (p. 795) Maximum length of user data in DDS_DataWriterQos (p. 1666) and DDS_PublicationBuiltinTopicData (p. 1905)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	reader_user_data_max_length (p. 1747)
	<< <i>extension</i> >> (p. 795) Maximum length of user data in DDS_DataReaderQos (p. 1611) and DDS_SubscriptionBuiltinTopicData (p. 2012)
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	max_partitions (p. 1748)
	<< <i>extension</i> >> (p. 795) Maximum number of partition name strings allowable in a DDS_PartitionQosPolicy (p. 1881).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	max_partition_cumulative_characters (p. 1748)
	<< <i>extension</i> >> (p. 795) Maximum number of combined characters allowable in all partition names in a DDS_PartitionQosPolicy (p. 1881).

DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	type_code_max_serialized_length (p. 1749)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Maximum size of serialized string for type code.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	type_object_max_serialized_length (p. 1749)
	<< <i>extension</i> >> (p. 795) The maximum length, in bytes, that the buffer to serialize a TypeObject can consume.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	type_object_max_deserialized_length (p. 1750)
	<< <i>extension</i> >> (p. 795) The maximum number of bytes that a deserialized TypeObject can consume.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	deserialized_type_object_dynamic_allocation_↔ threshold (p. 1751)
	<< <i>extension</i> >> (p. 795) A threshold, in bytes, for dynamic memory allocation for the deserialized Type↔Object.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	serialized_type_object_dynamic_allocation_↔ threshold (p. 1750)
	<< <i>extension</i> >> (p. 795) A threshold, in bytes, for dynamic memory allocation for the serialized TypeObject.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	contentfilter_property_max_length (p. 1751)
	<< <i>extension</i> >> (p. 795) This field is the maximum length of all data related to a Content-filtered topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	channel_seq_max_length (p. 1752)
	<< <i>extension</i> >> (p. 795) Maximum number of channels that can be specified in DDS_MultiChannelQos↔Policy (p. 1856) for MultiChannel DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	channel_filter_expression_max_length (p. 1752)
	<< <i>extension</i> >> (p. 795) Maximum length of a channel DDS_ChannelSettings_t::filter_expression (p. 1564) in a MultiChannel DataWriter.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	participant_property_list_max_length (p. 1752)
	<< <i>extension</i> >> (p. 795) Maximum number of properties associated with the DDS_DomainParticipant (p. 77).

DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	participant_property_string_max_length (p. 1753)
	<< <i>extension</i> >> (p. 795) Maximum string length of the properties associated with the DDS_DomainParticipant (p. 77).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	writer_property_list_max_length (p. 1753)
	<< <i>extension</i> >> (p. 795) Maximum number of properties associated with a DDS_DataWriter (p. 485).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	writer_property_string_max_length (p. 1753)
	<< <i>extension</i> >> (p. 795) Maximum string length of the properties associated with a DDS_DataWriter (p. 485).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	reader_property_list_max_length (p. 1754)
	<< <i>extension</i> >> (p. 795) Maximum number of properties associated with a DDS_DataReader (p. 615).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	reader_property_string_max_length (p. 1754)
	<< <i>extension</i> >> (p. 795) Maximum string length of the properties associated with a DDS_DataReader (p. 615).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	max_endpoint_groups (p. 1754)
	<< <i>extension</i> >> (p. 795) Maximum number of DDS_↵_EndpointGroup_t (p. 1784) allowable in a DDS_↵AvailabilityQosPolicy (p. 1547).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	max_endpoint_group_cumulative_characters (p. 1755)
	<< <i>extension</i> >> (p. 795) Maximum number of combined role_name characters allowed in all DDS_↵EndpointGroup_t (p. 1784) in a DDS_Availability_↵QosPolicy (p. 1547).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	transport_info_list_max_length (p. 1755)
	<< <i>extension</i> >> (p. 795) Maximum number of installed transports to send and receive information about in DDS_ParticipantBuiltinTopicData::transport_info (p. 1873).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	ignored_entity_replacement_kind (p. 1755)

	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Replacement policy for the ignored entities. It sets what entity can be replaced when resource limits set in DDS_DomainParticipantResourceLimits ↔ QosPolicy::ignored_entity_allocation (p. 1738) are reached.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_topic_query_allocation (p. 1756)
	<< <i>extension</i> >> (p. 795) Allocation settings applied to remote TopicQueries.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_topic_query_hash_buckets (p. 1756)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for remote TopicQueries.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	writer_data_tag_list_max_length (p. 1757)
	<< <i>extension</i> >> (p. 795) Maximum number of data tags associated with a DDS_DataWriter (p. 485).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	writer_data_tag_string_max_length (p. 1757)
	<< <i>extension</i> >> (p. 795) Maximum string length of the data tags associated with a DDS_DataWriter (p. 485).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	reader_data_tag_list_max_length (p. 1757)
	<< <i>extension</i> >> (p. 795) Maximum number of data tags associated with a DDS_DataReader (p. 615).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	reader_data_tag_string_max_length (p. 1758)
	<< <i>extension</i> >> (p. 795) Maximum string length of the data tags associated with a DDS_DataReader (p. 615).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	shm_ref_transfer_mode_max_segments (p. 1758)
	<< <i>extension</i> >> (p. 795) Maximum number of segments created by all DataWriters belonging to a DDS ↔ _DomainParticipant (p. 77).
DDS_EventQosPolicy (p. 1793)	thread (p. 1793)
	<< <i>extension</i> >> (p. 795) Event thread QoS.
DDS_EventQosPolicy (p. 1793)	initial_count (p. 1794)
	<< <i>extension</i> >> (p. 795) The initial number of events.
DDS_EventQosPolicy (p. 1793)	max_count (p. 1794)

	<< <i>extension</i> >> (p. 795) The maximum number of events.
DDS_DatabaseQosPolicy (p. 1581)	thread (p. 1582)
	<< <i>extension</i> >> (p. 795) Database thread settings.
DDS_DatabaseQosPolicy (p. 1581)	shutdown_timeout (p. 1583)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The maximum wait time during a shutdown.
DDS_DatabaseQosPolicy (p. 1581)	cleanup_period (p. 1583)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The period at which the database thread wakes up to remove deleted records.
DDS_DatabaseQosPolicy (p. 1581)	shutdown_cleanup_period (p. 1583)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The clean-up period used during database shut-down.
DDS_DatabaseQosPolicy (p. 1581)	initial_records (p. 1584)
	<< <i>extension</i> >> (p. 795) The initial number of total records.
DDS_DatabaseQosPolicy (p. 1581)	max_skiplist_level (p. 1584)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The maximum level of the skiplist.
DDS_DatabaseQosPolicy (p. 1581)	initial_weak_references (p. 1585)
	<< <i>extension</i> >> (p. 795) The initial number of weak references.
DDS_DatabaseQosPolicy (p. 1581)	max_weak_references (p. 1584)
	<< <i>extension</i> >> (p. 795) The maximum number of weak references.
DDS_ReceiverPoolQosPolicy (p. 1934)	thread (p. 1935)
	<< <i>extension</i> >> (p. 795) Receiver pool thread(s).
DDS_ReceiverPoolQosPolicy (p. 1934)	buffer_size (p. 1935)
	<< <i>extension</i> >> (p. 795) The receive buffer size in bytes.
DDS_ReceiverPoolQosPolicy (p. 1934)	buffer_alignment (p. 1936)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The receive buffer alignment.
DDS_DiscoveryConfigQosPolicy (p. 1689)	remote_participant_purge_kind (p. 1693)

	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.
DDS_DiscoveryConfigQosPolicy (p. 1689)	max_liveliness_loss_detection_period (p. 1694)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The maximum amount of time when a remote entity stops maintaining its liveliness and when the matched local entity realizes that fact.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_reader_resource_limits (p. 1696)
	<< <i>extension</i> >> (p. 795) Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 1689)	publication_reader (p. 1696)
	<< <i>extension</i> >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in publication reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	publication_reader_resource_limits (p. 1697)
	<< <i>extension</i> >> (p. 795) Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 1689)	subscription_reader (p. 1697)
	<< <i>extension</i> >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in subscription reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	subscription_reader_resource_limits (p. 1698)
	<< <i>extension</i> >> (p. 795) Resource limits.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_message_writer (p. 1702)
	<< <i>extension</i> >> (p. 795) 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. 1323) DDS_ReliabilityQos_↵PolicyKind (p. 1322).
DDS_DiscoveryConfigQosPolicy (p. 1689)	publication_writer_publish_mode (p. 1703)
	<< <i>extension</i> >> (p. 795) Publish mode policy for the built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	subscription_writer_publish_mode (p. 1704)
	<< <i>extension</i> >> (p. 795) Publish mode policy for the built-in subscription writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	asynchronous_publisher (p. 1704)

	<< <i>extension</i> >> (p. 795) Asynchronous publishing settings for the discovery DDS_Publisher (p. 443) and all entities that are created by it.
DDS_DiscoveryConfigQosPolicy (p. 1689)	default_domain_announcement_period (p. 1704)
	<< <i>extension</i> >> (p. 795) The period to announce a participant to the default domain 0.
DDS_DiscoveryConfigQosPolicy (p. 1689)	ignore_default_domain_announcements (p. 1705)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) 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. 1689)	participant_message_reader_reliability_kind (p. 1701)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Reliability policy for a built-in participant message reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_message_reader (p. 1702)
	<< <i>extension</i> >> (p. 795) 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. 1701) is set to DDS_RELIABLE_RELIABILITY_QOS (p. 1323).
DDS_DiscoveryConfigQosPolicy (p. 1689)	publication_writer (p. 1698)
	<< <i>extension</i> >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	publication_writer_data_lifecycle (p. 1699)
	<< <i>extension</i> >> (p. 795) Writer data lifecycle settings for a built-in publication writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	subscription_writer (p. 1700)
	<< <i>extension</i> >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in subscription writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	subscription_writer_data_lifecycle (p. 1700)
	<< <i>extension</i> >> (p. 795) Writer data lifecycle settings for a built-in subscription writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	builtin_discovery_plugins (p. 1701)
	<< <i>extension</i> >> (p. 795) Mask of built-in discovery plugin kinds.
DDS_DiscoveryConfigQosPolicy (p. 1689)	enabled_builtin_channels (p. 1701)

	<< <i>extension</i> >> (p. 795) The mask specifying which built-in channels should be enabled.
DDS_DiscoveryConfigQosPolicy (p. 1689)	service_request_writer (p. 1705)
	<< <i>extension</i> >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in DDS_ServiceRequest (p. 2001) writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	service_request_writer_data_lifecycle (p. 1706)
	<< <i>extension</i> >> (p. 795) Writer data lifecycle settings for a built-in DDS_ServiceRequest (p. 2001) writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	service_request_writer_publish_mode (p. 1706)
	<< <i>extension</i> >> (p. 795) Publish mode policy for the built-in service request writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	service_request_reader (p. 1707)
	<< <i>extension</i> >> (p. 795) RTPS reliable reader protocol-related configuration settings for a built-in DDS_ServiceRequest (p. 2001) reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	locator_reachability_assert_period (p. 1707)
	<< <i>extension</i> >> (p. 795) Period at which this DomainParticipant will assert the locators discovered from other DomainParticipants.
DDS_DiscoveryConfigQosPolicy (p. 1689)	locator_reachability_change_detection_period (p. 1708)
	<< <i>extension</i> >> (p. 795) Period at which this DomainParticipant will check if its locators are reachable from other DomainParticipants.
DDS_DiscoveryConfigQosPolicy (p. 1689)	secure_volatile_writer (p. 1709)
	<< <i>extension</i> >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in secure volatile writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	secure_volatile_writer_publish_mode (p. 1710)
	<< <i>extension</i> >> (p. 795) Publish mode policy for the built-in secure volatile writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	secure_volatile_reader (p. 1710)
	<< <i>extension</i> >> (p. 795) RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	endpoint_type_object_lb_serialization_threshold (p. 1711)
	<< <i>extension</i> >> (p. 795) Option to reduce the size required to propagate a TypeObject in Simple Endpoint Discovery

DDS_DiscoveryConfigQosPolicy (p. 1689)	dns_tracker_polling_period (p. 1711)
	<< extension >> (p. 795) 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. 1689)	participant_configuration_reader (p. 1714)
	<< extension >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a built-in participant configuration reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_configuration_reader_resource_limits (p. 1714)
	<< extension >> (p. 795) Resource limits for the built-in topic participant configuration reader.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_configuration_writer_publish_mode (p. 1712)
	<< extension >> (p. 795) Publish mode policy for the built-in participant configuration writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_configuration_writer (p. 1712)
	<< extension >> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a built-in participant configuration writer.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_configuration_writer_data_lifecycle (p. 1713)
	<< extension >> (p. 795) Writer data lifecycle settings for a built-in participant configuration writer.
DDS_RtpsReliableReaderProtocol_t (p. 1954)	min_heartbeat_response_delay (p. 1955)
	<< extension >> (p. 795) The minimum delay to respond to a heartbeat.
DDS_RtpsReliableReaderProtocol_t (p. 1954)	max_heartbeat_response_delay (p. 1955)
	<< extension >> (p. 795) The maximum delay to respond to a heartbeat.
DDS_RtpsReliableReaderProtocol_t (p. 1954)	heartbeat_suppression_duration (p. 1956)
	<< extension >> (p. 795) The duration a reader ignores consecutively received heartbeats
DDS_RtpsReliableReaderProtocol_t (p. 1954)	nack_period (p. 1956)
	<< extension >> (p. 795) The period at which to send NACKs
DDS_RtpsReliableReaderProtocol_t (p. 1954)	receive_window_size (p. 1957)
	<< extension >> (p. 795) The number of received out-of-order samples a reader can keep at a time

DDS_RtpsReliableReaderProtocol_t (p. 1954)	round_trip_time (p. 1957)
	<< <i>extension</i> >> (p. 795) The duration from sending a NACK to receiving a repair of a sample
DDS_RtpsReliableReaderProtocol_t (p. 1954)	app_ack_period (p. 1958)
	<< <i>extension</i> >> (p. 795) The period at which application-level acknowledgment messages are sent
DDS_RtpsReliableReaderProtocol_t (p. 1954)	min_app_ack_response_keep_duration (p. 1958)
	<< <i>extension</i> >> (p. 795) Minimum duration for which application-level acknowledgment response data is kept
DDS_RtpsReliableReaderProtocol_t (p. 1954)	samples_per_app_ack (p. 1959)
	<< <i>extension</i> >> (p. 795) The minimum number of samples acknowledged by one application-level acknowledgment message
DDS_RtpsReliableWriterProtocol_t (p. 1960)	low_watermark (p. 1961)
	<< <i>extension</i> >> (p. 795) 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. 1234) is considered to have changed.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	high_watermark (p. 1962)
	<< <i>extension</i> >> (p. 795) 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. 1234) is considered to have changed.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	virtual_heartbeat_period (p. 1965)
	<< <i>extension</i> >> (p. 795) 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. 1960)	samples_per_virtual_heartbeat (p. 1965)
	<< <i>extension</i> >> (p. 795) The number of samples that a reliable writer has to publish before sending a virtual heartbeat.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	max_heartbeat_retries (p. 1966)
	<< <i>extension</i> >> (p. 795) The maximum number of <i>periodic</i> heartbeat retries before marking a remote reader as inactive.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	inactivate_nonprogressing_readers (p. 1966)

	<< <i>extension</i> >> (p. 795) Whether to treat remote readers as inactive when their NACKs do not progress
DDS_RtpsReliableWriterProtocol_t (p. 1960)	heartbeats_per_max_samples (p. 1967)
	<< <i>extension</i> >> (p. 795) The number of piggyback heartbeats sent per max send window.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	min_nack_response_delay (p. 1968)
	<< <i>extension</i> >> (p. 795) The minimum delay to respond to a NACK or NACK_FRAG.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	max_nack_response_delay (p. 1968)
	<< <i>extension</i> >> (p. 795) The maximum delay to respond to a NACK or NACK_FRAG.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	nack_suppression_duration (p. 1969)
	<< <i>extension</i> >> (p. 795) The duration for ignoring consecutive NACKs/NACK_FRAGs that may trigger redundant repairs.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	max_bytes_per_nack_response (p. 1969)
	<< <i>extension</i> >> (p. 795) The maximum total message size when resending rejected samples or data fragments.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_enable_adaptive_sample_↔keep_duration (p. 1971)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Enables dynamic adjustment of sample keep duration in response to congestion.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_min_sample_keep_duration (p. 1970)
	<< <i>extension</i> >> (p. 795) The minimum duration a sample is queued for ACK-disabled readers.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_max_sample_keep_duration (p. 1971)
	<< <i>extension</i> >> (p. 795) The maximum duration a sample is queued for ACK-disabled readers.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_decrease_sample_keep_↔duration_factor (p. 1972)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Controls rate of contraction of dynamic sample keep duration.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_increase_sample_keep_↔duration_factor (p. 1972)

	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Controls rate of growth of dynamic sample keep duration.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	send_window_update_period (p. 1974)
	<< <i>extension</i> >> (p. 795) Period in which send window may be dynamically changed
DDS_RtpsReliableWriterProtocol_t (p. 1960)	send_window_increase_factor (p. 1975)
	<< <i>extension</i> >> (p. 795) Increases send window size by this percentage when reacting dynamically to network conditions
DDS_RtpsReliableWriterProtocol_t (p. 1960)	send_window_decrease_factor (p. 1976)
	<< <i>extension</i> >> (p. 795) Decreases send window size by this percentage when reacting dynamically to network conditions
DDS_RtpsReliableWriterProtocol_t (p. 1960)	enable_multicast_periodic_heartbeat (p. 1977)
	<< <i>extension</i> >> (p. 795) Whether periodic heartbeat messages are sent over multicast
DDS_RtpsReliableWriterProtocol_t (p. 1960)	multicast_resend_threshold (p. 1977)
	<< <i>extension</i> >> (p. 795) The minimum number of requesting readers needed to trigger a multicast resend
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_repair_piggyback_heartbeat (p. 1978)
	<< <i>extension</i> >> (p. 795) Prevents piggyback heartbeats from being sent with repair samples
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	initial_samples (p. 1558)
	<< <i>extension</i> >> (p. 795) Initial number of samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_samples (p. 1558)
	<< <i>extension</i> >> (p. 795) Maximum number of samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	initial_infos (p. 1559)
	<< <i>extension</i> >> (p. 795) Initial number of sample infos.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_infos (p. 1559)
	<< <i>extension</i> >> (p. 795) Maximum number of sample infos.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	initial_outstanding_reads (p. 1559)
	<< <i>extension</i> >> (p. 795) The initial number of outstanding reads that have not called finish yet on the same built-in topic DDS_DataReader (p. 615).
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_outstanding_reads (p. 1560)

	<< <i>extension</i> >> (p. 795) The maximum number of outstanding reads that have not called finish yet on the same built-in topic DDS_DataReader (p. 615).
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_samples_per_read (p. 1560)
	<< <i>extension</i> >> (p. 795) Maximum number of samples that can be read/taken on a same built-in topic DDS_DataReader (p. 615).
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	disable_fragmentation_support (p. 1560)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Determines whether the built-in topic DDS_DataReader (p. 615) can receive fragmented samples.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_fragmented_samples (p. 1561)
	<< <i>extension</i> >> (p. 795) The maximum number of samples for which the built-in topic DDS_DataReader (p. 615) may store fragments at a given point in time.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	initial_fragmented_samples (p. 1561)
	<< <i>extension</i> >> (p. 795) The initial number of samples for which a built-in topic DDS_DataReader (p. 615) may store fragments.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_fragmented_samples_per_remote_writer (p. 1562)
	<< <i>extension</i> >> (p. 795) The maximum number of samples per remote writer for which a built-in topic DDS_DataReader (p. 615) may store fragments.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	max_fragments_per_sample (p. 1562)
	<< <i>extension</i> >> (p. 795) Maximum number of fragments for a single sample.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	dynamically_allocate_fragmented_samples (p. 1563)
	<< <i>extension</i> >> (p. 795) Determines whether the built-in topic DDS_DataReader (p. 615) pre-allocates storage for storing fragmented samples.
DDS_DataTags (p. 1641)	tags (p. 1641)
	<< <i>spec</i> >> (p. 790) Sequence of data tags.
DDS_EntityNameQosPolicy (p. 1790)	role_name (p. 1791)
	<< <i>extension</i> >> (p. 795) The entity role name.
DDS_ServiceQosPolicy (p. 2000)	kind (p. 2000)
	<< <i>extension</i> >> (p. 795) The kind of service.
DDS_BatchQosPolicy (p. 1552)	max_flush_delay (p. 1554)

	<< extension >> (p. 795) The maximum flush delay.
DDS_BatchQosPolicy (p. 1552)	source_timestamp_resolution (p. 1554)
	<< extension >> (p. 795) Batch source timestamp resolution.
DDS_BatchQosPolicy (p. 1552)	thread_safe_write (p. 1555)
	<< extension >> (p. 795) Determines whether or not the write operation is thread safe.
DDS_ChannelSettings_t (p. 1564)	multicast_settings (p. 1564)
	<< extension >> (p. 795) A sequence of DDS_↵TransportMulticastSettings_t (p. 2060) used to configure the multicast addresses associated with a channel.
DDS_ChannelSettings_t (p. 1564)	filter_expression (p. 1564)
	<< extension >> (p. 795) A logical expression used to determine the data that will be published in the channel.
DDS_ChannelSettings_t (p. 1564)	priority (p. 1565)
	<< extension >> (p. 795) Publication priority
DDS_MultiChannelQosPolicy (p. 1856)	channels (p. 1858)
	<< extension >> (p. 795) A sequence of DDS_↵ChannelSettings_t (p. 1564) used to configure the channels' properties. If the length of the sequence is zero, the QoS policy will be ignored.
DDS_MultiChannelQosPolicy (p. 1856)	filter_name (p. 1858)
	<< extension >> (p. 795) Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.
DDS_TopicQueryDispatchQosPolicy (p. 2047)	enable (p. 2048)
	<< extension >> (p. 795) Allows this writer to dispatch TopicQueries
DDS_TopicQueryDispatchQosPolicy (p. 2047)	samples_per_period (p. 2049)
	<< extension >> (p. 795) Sets the maximum number of samples to publish in each publication_period.

5.79 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. 914) page.

Basic Settings

DDS_DeadlineQosPolicy (p. 1684)	period (p. 1685)
	<< <i>basic</i> >> (p. 783) << <i>spec</i> >> (p. 790) Duration of the deadline period.
DDS_OwnershipStrengthQosPolicy (p. 1868)	value (p. 1869)
	<< <i>basic</i> >> (p. 783) << <i>spec</i> >> (p. 790) The strength value used to arbitrate among multiple writers.
DDS_TimeBasedFilterQosPolicy (p. 2032)	minimum_separation (p. 2034)
	<< <i>basic</i> >> (p. 783) << <i>spec</i> >> (p. 790) The minimum separation duration between subsequent samples.
DDS_PartitionQosPolicy (p. 1881)	name (p. 1883)
	<< <i>basic</i> >> (p. 783) << <i>spec</i> >> (p. 790) A list of partition names
DDS_EntityFactoryQosPolicy (p. 1788)	autoenable_created_entities (p. 1789)
	<< <i>basic</i> >> (p. 783) << <i>spec</i> >> (p. 790) Specifies whether the entity acting as a factory automatically enables the instances it creates.
DDS_LifespanQosPolicy (p. 1815)	duration (p. 1816)
	<< <i>basic</i> >> (p. 783) << <i>spec</i> >> (p. 790) Maximum duration for the data's validity.
DDS_PropertyQosPolicy (p. 1901)	value (p. 1903)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) Sequence of properties.
DDS_LoggingQosPolicy (p. 1831)	verbosity (p. 1832)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) The verbirosities at which RTI Connnext diagnostic information is logged.
DDS_LoggingQosPolicy (p. 1831)	category (p. 1832)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) Categories of logged messages.
DDS_LoggingQosPolicy (p. 1831)	output_file (p. 1832)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) Specifies the file to which log messages will be redirected to.
DDS_LoggingQosPolicy (p. 1831)	output_file_suffix (p. 1833)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) Sets the file suffix when logging to a set of files

DDS_MonitoringEventDistributionSettings (p. 1840)	publication_period (p. 1843)
	<< basic >> (p. 783) << extension >> (p. 795) Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.
DDS_MonitoringPeriodicDistributionSettings (p. 1852)	polling_period (p. 1853)
	<< basic >> (p. 783) << extension >> (p. 795) Period at which the periodic metric thread polls and publishes the periodic metrics.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	max_historical_logs (p. 1845)
	<< basic >> (p. 783) << extension >> (p. 795) The number of log messages that RTI Monitoring Library 2.0 will keep as history.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	publication_period (p. 1846)
	<< basic >> (p. 783) << extension >> (p. 795) Period at which the logging thread publishes log messages.
DDS_MonitoringDedicatedParticipantSettings (p. 1836)	enable (p. 1836)
	<< basic >> (p. 783) << extension >> (p. 795) Enables the use of a dedicated DDS_DomainParticipant (p. 77) to distribute the RTI Connex application telemetry data.
DDS_MonitoringDedicatedParticipantSettings (p. 1836)	domain_id (p. 1837)
	<< basic >> (p. 783) << extension >> (p. 795) The domain ID used in the creation of RTI Monitoring Library 2.0 DDS_DomainParticipant (p. 77).
DDS_MonitoringDedicatedParticipantSettings (p. 1836)	collector_initial_peers (p. 1837)
	<< basic >> (p. 783) << extension >> (p. 795) 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. 1839).
DDS_MonitoringLoggingForwardingSettings (p. 1847)	middleware_forwarding_level (p. 1847)

	<< basic >> (p. 783) << extension >> (p. 795) Log messages with NDDS_CONFIG_LOG_FACILITY_↵ MIDDLEWARE (p. 1447) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1847)	security_event_forwarding_level (p. 1847)
	<< basic >> (p. 783) << extension >> (p. 795) Log messages with NDDS_CONFIG_LOG_FACILITY_↵ SECURITY_EVENT (p. 1447) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1847)	service_forwarding_level (p. 1848)
	<< basic >> (p. 783) << extension >> (p. 795) Log messages with NDDS_CONFIG_LOG_FACILITY_↵ SERVICE (p. 1447) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringLoggingForwardingSettings (p. 1847)	user_forwarding_level (p. 1848)
	<< basic >> (p. 783) << extension >> (p. 795) Log messages with NDDS_CONFIG_LOG_FACILITY_↵ USER (p. 1447) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.
DDS_MonitoringTelemetryData (p. 1855)	metrics (p. 1855)
	<< basic >> (p. 783) << extension >> (p. 795) Sequence of DDS_MonitoringMetricSelection (p. 1849) containing the event and periodic metrics that will be collected and distributed for a given set of observable resources.
DDS_MonitoringTelemetryData (p. 1855)	logs (p. 1856)
	<< basic >> (p. 783) << extension >> (p. 795) DDS_MonitoringLoggingForwardingSettings (p. 1847) containing the NDDS_Config_Syslog_↵ Verbosity (p. 1447) levels that will be forwarded for the different NDDS_Config_LogFacility (p. 1446).
DDS_MonitoringQosPolicy (p. 1853)	enable (p. 1854)
	<< basic >> (p. 783) << extension >> (p. 795) Enables the collection and distribution of telemetry data for an RTI Connext application using RTI Monitoring Library 2.0.
DDS_MonitoringQosPolicy (p. 1853)	application_name (p. 1854)

	<< basic >> (p. 783) << extension >> (p. 795) The name of the resource that represents this RTI Connex application.

Advanced Settings

DDS_UserDataQosPolicy (p. 2086)	value (p. 2087)
	<< spec >> (p. 790) a sequence of octets
DDS_GroupDataQosPolicy (p. 1803)	value (p. 1804)
	<< spec >> (p. 790) a sequence of octets
DDS_AvailabilityQosPolicy (p. 1547)	max_data_availability_waiting_time (p. 1550)
	<< extension >> (p. 795) Defines how much time to wait before delivering a sample to the application without having received some of the previous samples.
DDS_AvailabilityQosPolicy (p. 1547)	max_endpoint_availability_waiting_time (p. 1550)
	<< extension >> (p. 795) Defines how much time to wait to discover DataWriters providing samples for the same data source (virtual GUID).
DDS_AvailabilityQosPolicy (p. 1547)	required_matched_endpoint_groups (p. 1551)
	<< extension >> (p. 795) A sequence of endpoint groups.
DDS_LatencyBudgetQosPolicy (p. 1813)	duration (p. 1815)
	<< spec >> (p. 790) Duration of the maximum acceptable delay.
DDS_LoggingQosPolicy (p. 1831)	print_format (p. 1832)
	<< extension >> (p. 795) The format used to output RTI Connex diagnostic information.
DDS_LoggingQosPolicy (p. 1831)	max_bytes_per_file (p. 1833)
	<< extension >> (p. 795) Specifies the maximum number of bytes a single file can contain
DDS_LoggingQosPolicy (p. 1831)	max_files (p. 1834)
	<< extension >> (p. 795) Specifies the maximum number of files to create before overwriting the previous ones.
DDS_MonitoringEventDistributionSettings (p. 1840)	concurrency_level (p. 1841)
	<< extension >> (p. 795) Defines how concurrent the push of event metrics to RTI Monitoring Library 2.0 is.
DDS_MonitoringEventDistributionSettings (p. 1840)	datawriter_qos_profile_name (p. 1842)

	<< <i>extension</i> >> (p. 795) The fully qualified name of the profile used to configure the DDS_DataWriter (p. 485) that distributes event metrics.
DDS_MonitoringEventDistributionSettings (p. 1840)	thread (p. 1842)
	<< <i>extension</i> >> (p. 795) The settings of the event metric thread.
DDS_MonitoringPeriodicDistributionSettings (p. 1852)	datawriter_qos_profile_name (p. 1852)
	<< <i>extension</i> >> (p. 795) The fully qualified name of the profile used to configure the DDS_DataWriter (p. 485) that distributes periodic metrics.
DDS_MonitoringPeriodicDistributionSettings (p. 1852)	thread (p. 1852)
	<< <i>extension</i> >> (p. 795) The settings of the periodic metric thread.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	concurrency_level (p. 1844)
	<< <i>extension</i> >> (p. 795) Defines how concurrent the push of log messages to RTI Monitoring Library 2.0 is.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	datawriter_qos_profile_name (p. 1845)
	<< <i>extension</i> >> (p. 795) The fully qualified name of the profile used to configure the DDS_DataWriter (p. 485) that distributes log messages.
DDS_MonitoringLoggingDistributionSettings (p. 1843)	thread (p. 1846)
	<< <i>extension</i> >> (p. 795) The settings of the logging thread.
DDS_MonitoringDedicatedParticipantSettings (p. 1836)	participant_qos_profile_name (p. 1837)
	<< <i>extension</i> >> (p. 795) The fully qualified name of the profile used to configure the DDS_DomainParticipant (p. 77) that will be used to distribute telemetry data.
DDS_MonitoringDistributionSettings (p. 1838)	dedicated_participant (p. 1839)
	<< <i>extension</i> >> (p. 795) Configures the use of a dedicated DDS_DomainParticipant (p. 77) to distribute the RTI Connext application telemetry data.
DDS_MonitoringDistributionSettings (p. 1838)	publisher_qos_profile_name (p. 1839)

	<< extension >> (p. 795) The fully qualified name of the profile used to configure the Publishers that distribute telemetry data.
DDS_MonitoringDistributionSettings (p. 1838)	event_settings (p. 1839)
	<< extension >> (p. 795) Configures the distribution of event metrics.
DDS_MonitoringDistributionSettings (p. 1838)	periodic_settings (p. 1840)
	<< extension >> (p. 795) Configures the distribution of periodic metrics.
DDS_MonitoringDistributionSettings (p. 1838)	logging_settings (p. 1840)
	<< extension >> (p. 795) Configures the distribution of logging messages.
DDS_MonitoringMetricSelection (p. 1849)	resource_selection (p. 1849)
	<< extension >> (p. 795) An expression pattern that selects a subset of resources by matching the pattern to the resource names. DDS_MonitoringMetricSelection::enabled_metrics_selection (p. 1850) and DDS_MonitoringMetricSelection::disabled_metrics_selection (p. 1850) are applied to this subset of resources.
DDS_MonitoringMetricSelection (p. 1849)	enabled_metrics_selection (p. 1850)
	<< extension >> (p. 795) A sequence of POSIX fn-match 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. 1849).
DDS_MonitoringMetricSelection (p. 1849)	disabled_metrics_selection (p. 1850)
	<< extension >> (p. 795) A sequence of POSIX fn-match 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. 1849).
DDS_MonitoringQosPolicy (p. 1853)	distribution_settings (p. 1854)
	<< extension >> (p. 795) Configures the distribution of telemetry data.
DDS_MonitoringQosPolicy (p. 1853)	telemetry_data (p. 1855)
	<< extension >> (p. 795) Configures the telemetry data that will be distributed.

5.80 Other QoS Attributes

These attributes are characteristics of the QoS – for example if they are **mutable** (p. 949) or **immutable** (p. 914), or if they **apply per-instance** (p. 956).

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.

5.80.1 Detailed Description

These attributes are characteristics of the QoS – for example if they are **mutable** (p. 949) or **immutable** (p. 914), or if they **apply per-instance** (p. 956).

5.81 Applied Per-Instance

These QoS are applied per-instance.

These QoS are applied per-instance.

For example, the **DDS_HistoryQosPolicy::depth** (p. 1807) configures the number of samples that are stored in the DataWriter and DataReader queues *per-instance*.

Basic Settings

DDS_DurabilityQosPolicy (p. 1759)	writer_depth (p. 1762)
	<< basic >> (p. 783) << extension >> (p. 795) Indicates the number of samples per instance that a durable DDS_DataWriter (p. 485) will send to a late-joining DDS_DataReader (p. 615).
DDS_DeadlineQosPolicy (p. 1684)	period (p. 1685)
	<< basic >> (p. 783) << spec >> (p. 790) Duration of the deadline period.
DDS_OwnershipQosPolicy (p. 1863)	kind (p. 1868)
	<< basic >> (p. 783) << spec >> (p. 790) The kind of ownership
DDS_OwnershipStrengthQosPolicy (p. 1868)	value (p. 1869)

	<< basic >> (p. 783) << spec >> (p. 790) The strength value used to arbitrate among multiple writers.
DDS_TimeBasedFilterQosPolicy (p. 2032)	minimum_separation (p. 2034)
	<< basic >> (p. 783) << spec >> (p. 790) The minimum separation duration between subsequent samples.
DDS_DestinationOrderQosPolicy (p. 1686)	kind (p. 1688)
	<< basic >> (p. 783) << spec >> (p. 790) Specifies the desired kind of destination order.
DDS_HistoryQosPolicy (p. 1805)	kind (p. 1807)
	<< basic >> (p. 783) << spec >> (p. 790) Specifies the kind of history to be kept.
DDS_HistoryQosPolicy (p. 1805)	depth (p. 1807)
	<< basic >> (p. 783) << spec >> (p. 790) Specifies the number of samples per instance to be kept, when the kind is DDS_↵KEEP_LAST_HISTORY_QOS (p. 1293)

Advanced Settings

DDS_DurabilityServiceQosPolicy (p. 1764)	history_kind (p. 1765)
	<< spec >> (p. 790) The kind of history to apply in recouping durable data.
DDS_DurabilityServiceQosPolicy (p. 1764)	history_depth (p. 1765)
	<< spec >> (p. 790) Setting to use for the DDS_DurabilityQos↵Policy::writer_depth (p. 1762) when recouping durable data.
DDS_PresentationQosPolicy (p. 1890)	access_scope (p. 1893)
	<< spec >> (p. 790) Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
DDS_PresentationQosPolicy (p. 1890)	ordered_access (p. 1893)
	<< spec >> (p. 790) Specifies support for <i>ordered</i> access to the samples received at the subscription end. Controls whether ordered access is supported within the scope access_scope .
DDS_DestinationOrderQosPolicy (p. 1686)	scope (p. 1688)
	<< spec >> (p. 790) Specifies the desired scope of the source destination order.
DDS_DestinationOrderQosPolicy (p. 1686)	source_timestamp_tolerance (p. 1688)
	<< extension >> (p. 795) Allowed tolerance between source timestamps of consecutive samples.

5.82 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. 1870)

DDS_PublicationBuiltinTopicData (p. 1905)

DDS_SubscriptionBuiltinTopicData (p. 2012)

Basic Settings

DDS_DurabilityQosPolicy (p. 1759)	kind (p. 1762)
	<< <i>basic</i> >> (p. 783) << <i>spec</i> >> (p. 790) The kind of durability.
DDS_DataRepresentationQosPolicy (p. 1638)	compression_settings (p. 1640)
	<< <i>basic</i> >> (p. 783) << <i>extension</i> >> (p. 795) Structure that contains the compression settings.
DDS_DeadlineQosPolicy (p. 1684)	period (p. 1685)
	<< <i>basic</i> >> (p. 783) << <i>spec</i> >> (p. 790) Duration of the deadline period.
DDS_OwnershipQosPolicy (p. 1863)	kind (p. 1868)
	<< <i>basic</i> >> (p. 783) << <i>spec</i> >> (p. 790) The kind of ownership
DDS_OwnershipStrengthQosPolicy (p. 1868)	value (p. 1869)
	<< <i>basic</i> >> (p. 783) << <i>spec</i> >> (p. 790) The strength value used to arbitrate among multiple writers.
DDS_LivelinessQosPolicy (p. 1823)	kind (p. 1825)
	<< <i>basic</i> >> (p. 783) << <i>spec</i> >> (p. 790) The kind of liveliness desired.
DDS_LivelinessQosPolicy (p. 1823)	lease_duration (p. 1825)
	<< <i>basic</i> >> (p. 783) << <i>spec</i> >> (p. 790) The duration within which a DDS_DataWriter (p. 485) must be asserted, or else it is assumed to be not alive.
DDS_TimeBasedFilterQosPolicy (p. 2032)	minimum_separation (p. 2034)
	<< <i>basic</i> >> (p. 783) << <i>spec</i> >> (p. 790) The minimum separation duration between subsequent samples.
DDS_PartitionQosPolicy (p. 1881)	name (p. 1883)

	<< basic >> (p. 783) << spec >> (p. 790) A list of partition names
DDS_ReliabilityQosPolicy (p. 1936)	kind (p. 1939)
	<< basic >> (p. 783) << spec >> (p. 790) Kind of reliability.
DDS_ReliabilityQosPolicy (p. 1936)	max_blocking_time (p. 1939)
	<< basic >> (p. 783) << spec >> (p. 790) The maximum time a DataWriter can block on a write() call.
DDS_DestinationOrderQosPolicy (p. 1686)	kind (p. 1688)
	<< basic >> (p. 783) << spec >> (p. 790) Specifies the desired kind of destination order.
DDS_LifespanQosPolicy (p. 1815)	duration (p. 1816)
	<< basic >> (p. 783) << spec >> (p. 790) Maximum duration for the data's validity.
DDS_TransportMulticastQosPolicy (p. 2058)	value (p. 2059)
	<< basic >> (p. 783) << extension >> (p. 795) A sequence of multicast communications settings.
DDS_DiscoveryQosPolicy (p. 1715)	multicast_receive_addresses (p. 1717)
	<< basic >> (p. 783) << extension >> (p. 795) Specifies the multicast group addresses on which discovery-related meta-traffic can be received by the DomainParticipant.
DDS_DiscoveryConfigQosPolicy (p. 1689)	participant_liveliness_lease_duration (p. 1692)
	<< basic >> (p. 783) << extension >> (p. 795) The liveliness lease duration for the participant.
DDS_PropertyQosPolicy (p. 1901)	value (p. 1903)
	<< basic >> (p. 783) << extension >> (p. 795) Sequence of properties.
DDS_EntityNameQosPolicy (p. 1790)	name (p. 1790)
	<< basic >> (p. 783) << extension >> (p. 795) The name of the entity.

Advanced Settings

DDS_UserDataQosPolicy (p. 2086)	value (p. 2087)
	<< spec >> (p. 790) a sequence of octets
DDS_GroupDataQosPolicy (p. 1803)	value (p. 1804)
	<< spec >> (p. 790) a sequence of octets

DDS_DurabilityQosPolicy (p. 1759)	direct_communication (p. 1762)
	<< <i>extension</i> >> (p. 795) Indicates whether or not a TRANSIENT or PERSISTENT DDS_DataReader (p. 615) should receive samples directly from a TRANSIENT or PERSISTENT DDS_DataWriter (p. 485)
DDS_DurabilityServiceQosPolicy (p. 1764)	service_cleanup_delay (p. 1765)
	<< <i>spec</i> >> (p. 790) Controls when the service is able to remove all information regarding a data instances.
DDS_DurabilityServiceQosPolicy (p. 1764)	history_kind (p. 1765)
	<< <i>spec</i> >> (p. 790) The kind of history to apply in recouping durable data.
DDS_DurabilityServiceQosPolicy (p. 1764)	history_depth (p. 1765)
	<< <i>spec</i> >> (p. 790) Setting to use for the DDS_↔DurabilityQosPolicy::writer_depth (p. 1762) when recouping durable data.
DDS_DurabilityServiceQosPolicy (p. 1764)	max_samples (p. 1766)
	<< <i>spec</i> >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1764)	max_instances (p. 1766)
	<< <i>spec</i> >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_DurabilityServiceQosPolicy (p. 1764)	max_samples_per_instance (p. 1766)
	<< <i>spec</i> >> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
DDS_CompressionSettings_t (p. 1568)	compression_ids (p. 1568)
	<< <i>extension</i> >> (p. 795) Mask that represents the compression algorithms enabled.
DDS_DataRepresentationQosPolicy (p. 1638)	value (p. 1640)
	<< <i>spec</i> >> (p. 790) Sequence of representation identifiers.
DDS_PresentationQosPolicy (p. 1890)	access_scope (p. 1893)
	<< <i>spec</i> >> (p. 790) Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
DDS_PresentationQosPolicy (p. 1890)	coherent_access (p. 1893)
	<< <i>spec</i> >> (p. 790) Specifies support for <i>coherent</i> access. Controls whether coherent access is supported within the scope <i>access_scope</i> .
DDS_PresentationQosPolicy (p. 1890)	ordered_access (p. 1893)

	<< <i>spec</i> >> (p. 790) Specifies support for <i>ordered</i> access to the samples received at the subscription end. Controls whether ordered access is supported within the scope <code>access_scope</code> .
DDS_LatencyBudgetQosPolicy (p. 1813)	duration (p. 1815)
	<< <i>spec</i> >> (p. 790) Duration of the maximum acceptable delay.
DDS_ReliabilityQosPolicy (p. 1936)	acknowledgment_kind (p. 1939)
	<< <i>extension</i> >> (p. 795) Kind of reliable acknowledgment
DDS_ReliabilityQosPolicy (p. 1936)	instance_state_consistency_kind (p. 1940)
	<< <i>extension</i> >> (p. 795) Whether instance state consistency is enabled
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	kind (p. 2077)
	<< <i>spec</i> >> (p. 790) Type consistency kind.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	ignore_sequence_bounds (p. 2077)
	<< <i>spec</i> >> (p. 790) Controls whether sequence bounds are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	ignore_string_bounds (p. 2078)
	<< <i>spec</i> >> (p. 790) Controls whether string bounds are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	ignore_member_names (p. 2078)
	<< <i>spec</i> >> (p. 790) Controls whether member names are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	prevent_type_widening (p. 2078)
	<< <i>spec</i> >> (p. 790) Controls whether type widening is allowed.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	force_type_validation (p. 2079)
	<< <i>spec</i> >> (p. 790) Controls whether type information must be available in order to complete matching between a DDS_DataWriter (p. 485) and a DDS_DataReader (p. 615).
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	ignore_enum_literal_names (p. 2079)

	<< spec >> (p. 790) Controls whether enumeration constant names are taken into consideration for type assignability.
DDS_TransportUnicastQosPolicy (p. 2065)	value (p. 2067)
	<< extension >> (p. 795) A sequence of unicast communication settings.
DDS_DataReaderProtocolQosPolicy (p. 1596)	virtual_guid (p. 1597)
	<< extension >> (p. 795) The virtual GUID (Global Unique Identifier).
DDS_DataReaderProtocolQosPolicy (p. 1596)	expects_inline_qos (p. 1597)
	[DEPRECATED] (p. 965) << extension >> (p. 795) Specifies whether this DataReader expects inline QoS with every sample.
DDS_DataReaderProtocolQosPolicy (p. 1596)	disable_positive_acks (p. 1598)
	<< extension >> (p. 795) Whether the reader sends positive acknowledgements to writers.
DDS_DataWriterProtocolQosPolicy (p. 1649)	virtual_guid (p. 1650)
	<< extension >> (p. 795) The virtual GUID (Global Unique Identifier).
DDS_DataWriterProtocolQosPolicy (p. 1649)	disable_positive_acks (p. 1651)
	<< extension >> (p. 795) Controls whether or not the writer expects positive acknowledgements from matching readers.
DDS_DataTags (p. 1641)	tags (p. 1641)
	<< spec >> (p. 790) Sequence of data tags.
DDS_EntityNameQosPolicy (p. 1790)	role_name (p. 1791)
	<< extension >> (p. 795) The entity role name.
DDS_ServiceQosPolicy (p. 2000)	kind (p. 2000)
	<< extension >> (p. 795) The kind of service.
DDS_MultiChannelQosPolicy (p. 1856)	channels (p. 1858)
	<< extension >> (p. 795) A sequence of DDS_↵ChannelSettings_t (p. 1564) used to configure the channels' properties. If the length of the sequence is zero, the QoS policy will be ignored.
DDS_MultiChannelQosPolicy (p. 1856)	filter_name (p. 1858)
	<< extension >> (p. 795) Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.

DDS_TopicQueryDispatchQosPolicy (p. 2047)	enable (p. 2048)
	<< extension >> (p. 795) Allows this writer to dispatch TopicQueries

5.83 Matching

These QoS affect participant and endpoint matching.

These QoS affect participant and endpoint matching.

See the pages of each individual QoS for a description 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. 1245)) semantics.

Basic Settings

DDS_DurabilityQosPolicy (p. 1759)	kind (p. 1762)
	<< basic >> (p. 783) << spec >> (p. 790) The kind of durability.
DDS_DataRepresentationQosPolicy (p. 1638)	compression_settings (p. 1640)
	<< basic >> (p. 783) << extension >> (p. 795) Structure that contains the compression settings.
DDS_DeadlineQosPolicy (p. 1684)	period (p. 1685)
	<< basic >> (p. 783) << spec >> (p. 790) Duration of the deadline period.
DDS_OwnershipQosPolicy (p. 1863)	kind (p. 1868)
	<< basic >> (p. 783) << spec >> (p. 790) The kind of ownership
DDS_LivelinessQosPolicy (p. 1823)	kind (p. 1825)
	<< basic >> (p. 783) << spec >> (p. 790) The kind of liveliness desired.
DDS_LivelinessQosPolicy (p. 1823)	lease_duration (p. 1825)
	<< basic >> (p. 783) << spec >> (p. 790) The duration within which a DDS_DataWriter (p. 485) must be asserted, or else it is assumed to be not alive.
DDS_PartitionQosPolicy (p. 1881)	name (p. 1883)
	<< basic >> (p. 783) << spec >> (p. 790) A list of partition names
DDS_ReliabilityQosPolicy (p. 1936)	kind (p. 1939)
	<< basic >> (p. 783) << spec >> (p. 790) Kind of reliability.

DDS_DestinationOrderQosPolicy (p. 1686)	kind (p. 1688)
	<< basic >> (p. 783) << spec >> (p. 790) Specifies the desired kind of destination order.

Advanced Settings

DDS_CompressionSettings_t (p. 1568)	compression_ids (p. 1568)
	<< extension >> (p. 795) Mask that represents the compression algorithms enabled.
DDS_DataRepresentationQosPolicy (p. 1638)	value (p. 1640)
	<< spec >> (p. 790) Sequence of representation identifiers.
DDS_PresentationQosPolicy (p. 1890)	access_scope (p. 1893)
	<< spec >> (p. 790) Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
DDS_PresentationQosPolicy (p. 1890)	coherent_access (p. 1893)
	<< spec >> (p. 790) Specifies support for <i>coherent</i> access. Controls whether coherent access is supported within the scope <code>access_scope</code> .
DDS_PresentationQosPolicy (p. 1890)	ordered_access (p. 1893)
	<< spec >> (p. 790) Specifies support for <i>ordered</i> access to the samples received at the subscription end. Controls whether ordered access is supported within the scope <code>access_scope</code> .
DDS_LatencyBudgetQosPolicy (p. 1813)	duration (p. 1815)
	<< spec >> (p. 790) Duration of the maximum acceptable delay.
DDS_ReliabilityQosPolicy (p. 1936)	acknowledgment_kind (p. 1939)
	<< extension >> (p. 795) Kind of reliable acknowledgment
DDS_ReliabilityQosPolicy (p. 1936)	instance_state_consistency_kind (p. 1940)
	<< extension >> (p. 795) Whether instance state consistency is enabled
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	kind (p. 2077)
	<< spec >> (p. 790) Type consistency kind.

DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	ignore_sequence_bounds (p. 2077)
	<< <i>spec</i> >> (p. 790) Controls whether sequence bounds are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	ignore_string_bounds (p. 2078)
	<< <i>spec</i> >> (p. 790) Controls whether string bounds are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	ignore_member_names (p. 2078)
	<< <i>spec</i> >> (p. 790) Controls whether member names are taken into consideration for type assignability.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	prevent_type_widening (p. 2078)
	<< <i>spec</i> >> (p. 790) Controls whether type widening is allowed.
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	force_type_validation (p. 2079)
	<< <i>spec</i> >> (p. 790) Controls whether type information must be available in order to complete matching between a DDS_DataWriter (p. 485) and a DDS_DataReader (p. 615).
DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)	ignore_enum_literal_names (p. 2079)
	<< <i>spec</i> >> (p. 790) Controls whether enumeration constant names are taken into consideration for type assignability.

5.84 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. 1949)	instance_hash_buckets (p. 1953)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for instances.

DDS_TypeSupportQosPolicy (p. 2081)	plugin_data (p. 2082)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Value to pass into the type plugin's de-/serialization function
DDS_TypeSupportQosPolicy (p. 2081)	cdr_padding_kind (p. 2082)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Determines whether or not the padding bytes will be set to zero during CDR serialization.
DDS_AsymynchronousPublisherQosPolicy (p. 1539)	disable_asynchronous_write (p. 1540)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Disable asynchronous publishing.
DDS_AsymynchronousPublisherQosPolicy (p. 1539)	disable_asynchronous_batch (p. 1541)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Disable asynchronous batch flushing.
DDS_AsymynchronousPublisherQosPolicy (p. 1539)	disable_topic_query_publication (p. 1542)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Disable topic query publication.
DDS_DiscoveryQosPolicy (p. 1715)	enable_endpoint_discovery (p. 1718)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Whether to automatically enable endpoint discovery for all the remote participants.
DDS_DataReaderResourceLimitsQosPolicy (p. 1619)	disable_fragmentation_support (p. 1626)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Determines whether the DDS_DataReader (p. 615) can receive fragmented samples.
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	initial_concurrent_blocking_threads (p. 1675)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The initial number of threads that are allowed to concurrently block on write call on the same DDS_DataWriter (p. 485).
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	max_concurrent_blocking_threads (p. 1675)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The maximum number of threads that are allowed to concurrently block on write call on the same DDS_DataWriter (p. 485).
DDS_DataWriterResourceLimitsQosPolicy (p. 1673)	autoregister_instances (p. 1678)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Whether or not to automatically register new instances
DDS_DataReaderProtocolQosPolicy (p. 1596)	expects_inline_qos (p. 1597)

	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Specifies whether this DataReader expects inline QoS with every sample.
DDS_SystemResourceLimitsQosPolicy (p. 2024)	max_objects_per_thread (p. 2025)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The maximum number of objects that can be stored per thread for a DDS_DomainParticipantFactory (p. 32).
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_writer_hash_buckets (p. 1741)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Hash_Buckets settings applied to local DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_reader_hash_buckets (p. 1741)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for local DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_publisher_hash_buckets (p. 1741)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for local Publisher.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_subscriber_hash_buckets (p. 1742)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for local Subscriber.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	local_topic_hash_buckets (p. 1742)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for local Topic.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_writer_hash_buckets (p. 1742)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for remote DataWriters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_reader_hash_buckets (p. 1743)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for remote DataReaders.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_participant_hash_buckets (p. 1743)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for remote DomainParticipants.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	matching_writer_reader_pair_hash_buckets (p. 1743)

	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for matching local writer and remote/local reader pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	matching_reader_writer_pair_hash_buckets (p. 1744)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for matching local reader and remote/local writer pairs.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	ignored_entity_hash_buckets (p. 1744)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for ignored entities.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	content_filtered_topic_hash_buckets (p. 1744)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for content filtered topics.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	content_filter_hash_buckets (p. 1745)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for content filters.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	flow_controller_hash_buckets (p. 1745)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for flow controllers.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	type_code_max_serialized_length (p. 1749)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Maximum size of serialized string for type code.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	ignored_entity_replacement_kind (p. 1755)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Replacement policy for the ignored entities. It sets what entity can be replaced when resource limits set in DDS_DomainParticipantResourceLimitsQosPolicy::ignored_entity_allocation (p. 1738) are reached.
DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)	remote_topic_query_hash_buckets (p. 1756)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Number of hash buckets for remote TopicQueries.
DDS_DatabaseQosPolicy (p. 1581)	shutdown_timeout (p. 1583)

	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The maximum wait time during a shutdown.
DDS_DatabaseQosPolicy (p. 1581)	cleanup_period (p. 1583)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The period at which the database thread wakes up to remove deleted records.
DDS_DatabaseQosPolicy (p. 1581)	shutdown_cleanup_period (p. 1583)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The clean-up period used during database shut-down.
DDS_DatabaseQosPolicy (p. 1581)	max_skiplist_level (p. 1584)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The maximum level of the skiplist.
DDS_ReceiverPoolQosPolicy (p. 1934)	buffer_alignment (p. 1936)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The receive buffer alignment.
DDS_DiscoveryConfigQosPolicy (p. 1689)	remote_participant_purge_kind (p. 1693)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.
DDS_DiscoveryConfigQosPolicy (p. 1689)	max_liveliness_loss_detection_period (p. 1694)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) 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. 1689)	ignore_default_domain_announcements (p. 1705)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) 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. 1689)	participant_message_reader_reliability_kind (p. 1701)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Reliability policy for a built-in participant message reader.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_enable_adaptive_sample_↔keep_duration (p. 1971)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Enables dynamic adjustment of sample keep duration in response to congestion.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_decrease_sample_keep_↔duration_factor (p. 1972)

	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Controls rate of contraction of dynamic sample keep duration.
DDS_RtpsReliableWriterProtocol_t (p. 1960)	disable_positive_acks_increase_sample_keep_duration_factor (p. 1972)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Controls rate of growth of dynamic sample keep duration.
DDS_BuiltinTopicReaderResourceLimits_t (p. 1557)	disable_fragmentation_support (p. 1560)
	[DEPRECATED] (p. 965) << <i>extension</i> >> (p. 795) Determines whether the built-in topic DDS_DataReader (p. 615) can receive fragmented samples.

5.85 System Properties

System Properties.

System Properties.

RTI Connext uses the **DDS_PropertyQosPolicy** (p. 1901) of a DomainParticipant to maintain a set of properties that provide system information such as hostname.

Unless the default **DDS_DomainParticipantQos** (p. 1726) value is overwritten, the system properties are automatically set in the **DDS_DomainParticipantQos** (p. 1726) obtained by calling the function **DDS_DomainParticipantFactory**↵
_get_default_participant_qos (p. 41) or using the constant **DDS_PARTICIPANT_QOS_DEFAULT** (p. 65).

System properties are also automatically set in the **DDS_DomainParticipantQos** (p. 1726) 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. 1871).

You can disable the propagation of individual properties by setting the flag **DDS_Property_t::propagate** (p. 1901) to **DDS_BOOLEAN_FALSE** (p. 1201) or by removing the property using the function **DDS_PropertyQosPolicyHelper**↵
_remove_property (p. 1314).

The number of system properties set on the **DDS_DomainParticipantQos** (p. 1726) is platform specific.

5.85.1 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

5.85.2 System Resource Consideration

System properties are affected by the resource limits `DDS_DomainParticipantResourceLimitsQosPolicy::participant_property_list_max_length` (p. 1752) and `DDS_DomainParticipantResourceLimitsQosPolicy::participant_property_string_max_length` (p. 1753).

5.86 Configuring QoS Profiles with XML

APIs related to XML QoS Profiles.

APIs related to XML QoS Profiles.

5.86.1 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:

- `DDS_DomainParticipantFactory_create_participant` (p. 41)
- `DDS_DomainParticipantFactory_create_participant_with_profile` (p. 43)
- `DDS_DomainParticipantFactory_set_default_participant_qos_with_profile` (p. 40)
- `DDS_DomainParticipantFactory_get_default_participant_qos` (p. 41)
- `DDS_DomainParticipantFactory_set_default_library` (p. 48)
- `DDS_DomainParticipantFactory_set_default_profile` (p. 49)
- `DDS_DomainParticipantFactory_get_participant_qos_from_profile` (p. 52)
- `DDS_DomainParticipantFactory_get_topic_qos_from_profile` (p. 56)
- `DDS_DomainParticipantFactory_get_topic_qos_from_profile_w_topic_name` (p. 57)

- **DDS_DomainParticipantFactory_get_publisher_qos_from_profile** (p. 52)
- **DDS_DomainParticipantFactory_get_subscriber_qos_from_profile** (p. 53)
- **DDS_DomainParticipantFactory_get_datawriter_qos_from_profile** (p. 55)
- **DDS_DomainParticipantFactory_get_datawriter_qos_from_profile_w_topic_name** (p. 55)
- **DDS_DomainParticipantFactory_get_datareader_qos_from_profile** (p. 54)
- **DDS_DomainParticipantFactory_get_datareader_qos_from_profile_w_topic_name** (p. 54)
- **DDS_DomainParticipantFactory_get_qos_profile_libraries** (p. 57)
- **DDS_DomainParticipantFactory_get_qos_profiles** (p. 58)
- **DDS_DomainParticipantFactory_load_profiles** (p. 46)

The QoS profiles are reloaded replacing previously loaded profiles when the following operations are called:

- **DDS_DomainParticipantFactory_set_qos** (p. 46)
- **DDS_DomainParticipantFactory_reload_profiles** (p. 47)

The **DDS_DomainParticipantFactory_unload_profiles** (p. 47) 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 **NDDDS_QOS_PROFILES.xml** in **\$NDDSHOME/resource/xml** is loaded if it exists and **DDS_ProfileQosPolicy::ignore_resource_profile** (p. 1900) in **DDS_ProfileQosPolicy** (p. 1898) is set to **DDS_BOOLEAN_FALSE** (p. 1201) (first to be loaded). An example file, **NDDDS_QOS_PROFILES.example.xml**, is available for reference.
- The URL groups separated by semicolons referenced by the environment variable **NDDDS_QOS_PROFILES** are loaded if they exist and **DDS_ProfileQosPolicy::ignore_environment_profile** (p. 1899) in **DDS_ProfileQosPolicy** (p. 1898) is set to **DDS_BOOLEAN_FALSE** (p. 1201).
- The file **USER_QOS_PROFILES.xml** in the working directory will be loaded if it exists and **DDS_ProfileQosPolicy::ignore_user_profile** (p. 1899) in **DDS_ProfileQosPolicy** (p. 1898) is set to **DDS_BOOLEAN_FALSE** (p. 1201).
- The URL groups referenced by **DDS_ProfileQosPolicy::url_profile** (p. 1899) in **DDS_ProfileQosPolicy** (p. 1898) will be loaded if specified.
- The sequence of XML strings referenced by **DDS_ProfileQosPolicy::string_profile** (p. 1899) will be loaded if specified (last to be loaded).

The above methods can be combined together.

5.86.2 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> . . . </qos_library></dds>"`

If the URL schema name is omitted, RTI Connext will assume a file name. For example:

File Specification: `/usr/local/default_dds.xml`

5.86.2.1 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.

5.86.2.2 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. 1899) is set to `DDS_BOOLEAN_FALSE` (p. 1201)

5.86.2.3 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. 1394).

For more information on XML Configuration, refer to the `User's Manual`.

5.87 Publication Example

A data publication example.

A data publication example.

5.87.1 A typical publication example

Prep

- Create user data types using `rtiddsgen`. See the `Code Generator User's Manual`.

Set up

- **Get the factory** (p. 976)
- **Set up participant** (p. 976)
- **Set up publisher** (p. 981)
- **Register user data type(s)** (p. 978)
- **Set up topic(s)** (p. 978)
- **Set up data writer(s)** (p. 982)

Adjust the desired quality of service (QoS)

- **Adjust QoS on entities as necessary** (p. 989)

Send data

- **Send data** (p. 983)

Tear down

- **Tear down data writer(s)** (p. 983)
- **Tear down topic(s)** (p. 978)
- **Tear down publisher** (p. 981)
- **Tear down participant** (p. 977)

5.88 Subscription Example

A data subscription example.

A data subscription example.

5.88.1 A typical subscription example

Prep

- Create user data types using `rtiddsgen`. See the `Code Generator User's Manual`.

Set up

- **Get the factory** (p. 976)
- **Set up participant** (p. 976)
- **Set up subscriber** (p. 983)
- **Register user data type(s)** (p. 978)
- **Set up topic(s)** (p. 978)
- **Set up data reader(s)** (p. 986)

- **Set up data reader** (p. 986) **OR** **Set up subscriber** (p. 984) to receive data

Adjust the desired quality of service (QoS)

- **Adjust QoS on entities as necessary** (p. 989)

Receive data

- Access received data either **via a reader** (p. 987) **OR via a subscriber** (p. 984) (possibly in a **ordered or coherent** (p. 985) manner)

Tear down

- **Tear down data reader(s)** (p. 988)
- **Tear down topic(s)** (p. 978)
- **Tear down subscriber** (p. 985)
- **Tear down participant** (p. 977)

5.89 Participant Use Cases

Working with domain participants.

Working with domain participants.

5.89.1 Turning off auto-enable of newly created participant(s)

- Get the factory (p. 976)
- Change the value of the **ENTITY_FACTORY** (p. 1289) for the **DDS_DomainParticipantFactory** (p. 32)


```
struct DDS_DomainParticipantFactoryQos factory_qos = DDS_DomainParticipantFactoryQos_INITIALIZER;
if (DDS_DomainParticipantFactory_get_qos(factory, &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 (DDS_DomainParticipantFactory_set_qos(factory, &factory_qos)
    != DDS_RETCODE_OK) {
    printf("***Error: failed to set domain participant factory qos\n");
}

DDS_DomainParticipantFactoryQos_finalize(&factory_qos);
```

5.89.2 Getting the factory

- Get the DDS_DomainParticipantFactory instance:


```
DDS_DomainParticipantFactory* factory = NULL;
factory = DDS_DomainParticipantFactory_get_instance();
if (factory == NULL) {
    printf("***Error: failed to get domain participant factory\n");
}
```

5.89.3 Setting up a participant

- Get the factory (p. 976)
- Create DDS_DomainParticipant:


```
struct DDS_DomainParticipantQos participant_qos =
    DDS_DomainParticipantQos_INITIALIZER;
DDS_DomainParticipant* participant;
struct DDS_DomainParticipantListener participant_listener =
    DDS_DomainParticipantListener_INITIALIZER;
DDS_ReturnCode_t retcode;
/* Set the initial peers. These list 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 addresses 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 long NDDS_DISCOVERY_INITIAL_PEERS_LENGTH =
    sizeof(NDDS_DISCOVERY_INITIAL_PEERS)/sizeof(const char*);

/* MyDomainParticipantListener_* functions are user defined to match
   DDS_DomainParticipantListener functions */
participant_listener.as_topiclistener.on_inconsistent_topic =
    MyDomainParticipantListener_InconsistentTopic;
participant_listener.as_publisherlistener.as_datawriterlistener.on_offered_deadline_missed =
```



```

    MyDomainParticipantListener_OfferedDeadlineMissed;
    participant_listener.as_publisherlistener.as_datawriterlistener.on_offered_incompatible_qos =
        MyDomainParticipantListener_OfferedIncompatibleQos;
    participant_listener.as_publisherlistener.as_datawriterlistener.on_liveliness_lost =
        MyDomainParticipantListener_LivelinessLost;
    participant_listener.as_publisherlistener.as_datawriterlistener.on_publication_matched =
        MyDomainParticipantListener_PublicationMatch;
    participant_listener.as_subscriberlistener.on_data_on_readers =
        MyDomainParticipantListener_DataOnReaders;
    participant_listener.as_subscriberlistener.as_datareaderlistener.on_requested_deadline_missed =
        MyDomainParticipantListener_RequestedDeadlineMissed;
    participant_listener.as_subscriberlistener.as_datareaderlistener.on_requested_incompatible_qos =
        MyDomainParticipantListener_RequestedIncompatibleQos;
    participant_listener.as_subscriberlistener.as_datareaderlistener.on_sample_rejected =
        MyDomainParticipantListener_SampleRejected;
    participant_listener.as_subscriberlistener.as_datareaderlistener.on_liveliness_changed =
        MyDomainParticipantListener_LivelinessChanged;
    participant_listener.as_subscriberlistener.as_datareaderlistener.on_data_available =
        MyDomainParticipantListener_DataAvailable;
    participant_listener.as_subscriberlistener.as_datareaderlistener.on_subscription_matched =
        MyDomainParticipantListener_SubscriptionMatched;
    participant_listener.as_subscriberlistener.as_datareaderlistener.on_sample_lost =
        MyDomainParticipantListener_SampleLost;
    /* initialize participant_qos with default values */
    retcode = DDS_DomainParticipantFactory_get_default_participant_qos(factory,
                                                                    &participant_qos);

    if (retcode != DDS_RETCODE_OK) {
        printf("***Error: failed to get default participant qos\n");
    }
    if (!DDS_StringSeq_from_array(&participant_qos.discovery.initial_peers,
                                NDDS_DISCOVERY_INITIAL_PEERS,
                                NDDS_DISCOVERY_INITIAL_PEERS_LENGTH)) {
        printf("***Error: failed to set discovery.initial_peers qos\n");
    }

    /* Create the participant */
    participant =
        DDS_DomainParticipantFactory_create_participant(factory,
                                                        domain_id,
                                                        &participant_qos,
                                                        &participant_listener /* or NULL */,
                                                        DDS_STATUS_MASK_ALL);

    if (participant == NULL) {
        printf("***Error: failed to create domain participant\n");
    }
    return participant;

```

5.89.4 Tearing down a participant

- Get the factory (p. 976)
- Delete DDS_DomainParticipant:

```

DDS_ReturnCode_t retcode;
retcode = DDS_DomainParticipantFactory_delete_participant(
    factory, participant);
if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to delete domain participant\n");
}

```

5.90 Topic Use Cases

Working with topics.

Working with topics.

5.90.1 Registering a user data type

- Set up participant (p. 976)
- 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) {
    printf("***Error: failed to register type\n");
}
```

5.90.2 Setting up a topic

- Set up participant (p. 976)
- Ensure user data type is registered (p. 978)
- Create a DDS_Topic under the name "my_topic"


```
const char* topic_name = "my_topic";
const char* type_name = "My_Type"; /* user data type */
struct DDS_TopicQos topic_qos = DDS_TopicQos_INITIALIZER;
DDS_Topic* topic;
struct DDS_TopicListener topic_listener = DDS_TopicListener_INITIALIZER;
/* MyTopicListener_InconsistentTopic function is user defined to match
   DDS_TopicListener_InconsistentTopicCallback */
topic_listener.on_inconsistent_topic = MyTopicListener_InconsistentTopic;
retcode = DDS_DomainParticipant_get_default_topic_qos(participant,
                                                       &topic_qos);

if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to get default topic qos\n");
}
topic = DDS_DomainParticipant_create_topic(participant,
                                           topic_name,
                                           type_name,
                                           &topic_qos,
                                           &topic_listener /* or NULL */,
                                           DDS_STATUS_MASK_ALL);

if (topic == NULL) {
    printf("***Error: failed to create topic\n");
}
```

5.90.3 Tearing down a topic

- Delete DDS_Topic:


```
DDS_ReturnCode_t retcode;
retcode = DDS_DomainParticipant_delete_topic(participant, topic);
if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to delete topic\n");
}
```

5.91 FlowController Use Cases

Working with flow controllers.

Working with flow controllers.

5.91.1 Creating a flow controller

- Set up participant (p. 976)

- Create a flow controller

```
DDS_ReturnCode_t retcode;
DDS_FlowController *controller = NULL;
struct DDS_FlowControllerProperty_t property = DDS_FlowControllerProperty_t_INITIALIZER;
retcode = DDS_DomainParticipant_get_default_flowcontroller_property(
    participant, &property);
if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to get default flow controller property\n");
}
/* optionally modify flow controller property values */
controller = DDS_DomainParticipant_create_flowcontroller(
    participant, "my flow controller name", &property);
if (controller == NULL) {
    printf("***Error: failed to create flow controller\n");
}
```

5.91.2 Flow controlling a data writer

- Set up participant (p. 976)

- Create flow controller (p. 979)

- Create an asynchronous data writer, **FooDataWriter** (p. 2113), of user data type **Foo** (p. 2109) :

```
struct DDS_DataWriterQos writer_qos = DDS_DataWriterQos_INITIALIZER;
DDS_DataWriter* writer;
struct DDS_DataWriterListener writer_listener =
    DDS_DataWriterListener_INITIALIZER;
/* MyWriterListener_* functions are user defined to match
   DDS_DataWriterListener functions */
writer_listener.on_offered_deadline_missed =
    MyWriterListener_OfferedDeadlineMissed;
writer_listener.on_offered_incompatible_qos =
    MyWriterListener_OfferedIncompatibleQos;
writer_listener.on_liveliness_lost = MyWriterListener_LivelinessLost;
writer_listener.on_publication_matched = MyWriterListener_PublicationMatch;
retcode = DDS_Publisher_get_default_datawriter_qos(publisher, &writer_qos);
if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to get default datawriter qos\n");
}
/* Change the writer QoS to publish asynchronously */
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;
writer = DDS_Publisher_create_datawriter(publisher,
    topic,
    &writer_qos,
    &writer_listener /* or NULL */,
    DDS_STATUS_MASK_ALL);
if (writer == NULL) {
    printf("***Error: failed to create writer\n");
}
/* Send data asynchronously... */
/* Wait for asynchronous send completes, if desired */
retcode = DDS_DataWriter_wait_for_asynchronous_publishing(writer, &timeout);
if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to wait for asynchronous publishing\n");
}
```

5.91.3 Using the built-in flow controllers

RTI Connex provides several built-in flow controllers.

The **DDS_DEFAULT_FLOW_CONTROLLER_NAME** (p. 564) built-in flow controller provides the basic asynchronous writer behavior. When calling **FooDataWriter_write** (p. 496), the call signals the **DDS_Publisher** (p. 443) asynchronous publishing thread (**DDS_PublisherQos::asynchronous_publisher** (p. 1919)) to send the actual data. As with any **DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 1319) **DDS_DataWriter** (p. 485), the **FooDataWriter_write** (p. 496) call returns immediately afterwards. The data is sent immediately in the context of the **DDS_Publisher** (p. 443) asynchronous publishing thread.

When using the **DDS_FIXED_RATE_FLOW_CONTROLLER_NAME** (p. 564) flow controller, data is also sent in the context of the **DDS_Publisher** (p. 443) asynchronous publishing thread, but at a regular fixed interval. The thread accumulates samples from different **DDS_DataWriter** (p. 485) instances and generates data on the wire only once per **DDS_FlowControllerTokenBucketProperty_t::period** (p. 1802).

In contrast, the **DDS_ON_DEMAND_FLOW_CONTROLLER_NAME** (p. 565) flow controller permits flow only when **DDS_FlowController_trigger_flow** (p. 563) is called. The data is still sent in the context of the **DDS_Publisher** (p. 443) asynchronous publishing thread. The thread accumulates samples from different **DDS_DataWriter** (p. 485) instances (across any **DDS_Publisher** (p. 443)) and sends all data since the previous trigger.

The properties of the built-in **DDS_FlowController** (p. 559) instances can be adjusted.

- **Set up participant** (p. 976)

- **Lookup built-in flow controller**

```
DDS_FlowController *controller = NULL;
controller = DDS_DomainParticipant_lookup_flowcontroller(
    participant, 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;
struct DDS_FlowControllerProperty_t property = DDS_FlowControllerProperty_t_INITIALIZER;
/* Get the property of the flow controller */
retcode = DDS_FlowController_get_property(controller, &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 = DDS_FlowController_set_property(controller, &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. 979)

5.91.4 Shaping the network traffic for a particular transport

- **Set up participant** (p. 976)
- **Create the transports** (p. 992)
- **Create a separate flow controller for each transport** (p. 979)
- Configure **DDS_DataWriter** (p. 485) instances to only use a single transport
- **Associate all data writers using the same transport to the corresponding flow controller** (p. 979)
- For each transport, the corresponding flow controller limits the network traffic based on the token bucket properties

5.91.5 Coalescing multiple samples in a single network packet

- **Set up participant** (p. 976)
- **Create a flow controller with a desired token bucket period** (p. 979)
- **Associate the data writer with the flow controller** (p. 979)
- 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).

5.92 Publisher Use Cases

Working with publishers.

Working with publishers.

5.92.1 Setting up a publisher

- **Set up participant** (p. 976)
- **Create a DDS_Publisher**

```

struct DDS_PublisherQos publisher_qos = DDS_PublisherQos_INITIALIZER;
DDS_Publisher* publisher;
struct DDS_PublisherListener publisher_listener =
    DDS_PublisherListener_INITIALIZER;
DDS_ReturnCode_t retcode;
/* MyPublisherListener_* functions are user defined to match
   DDS_PublisherListener functions */
publisher_listener.as_datawriterlistener.on_offered_deadline_missed =
    MyPublisherListener_OfferedDeadlineMissed;
publisher_listener.as_datawriterlistener.on_offered_incompatible_qos =
    MyPublisherListener_OfferedIncompatibleQos;
publisher_listener.as_datawriterlistener.on_liveliness_lost =
    MyPublisherListener_LivelinessLost;
publisher_listener.as_datawriterlistener.on_publication_matched =
    MyPublisherListener_PublicationMatch;
retcode = DDS_DomainParticipant_get_default_publisher_qos(participant,
    &publisher_qos);

if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to get default publisher qos\n");
}
publisher = DDS_DomainParticipant_create_publisher(participant,
    &publisher_qos,
    &publisher_listener /* or NULL */,
    DDS_STATUS_MASK_ALL);

if (publisher == NULL) {
    printf("***Error: failed to create publisher\n");
}

```

5.92.2 Tearing down a publisher

- **Delete DDS_Publisher:**

```

DDS_ReturnCode_t retcode;
retcode = DDS_DomainParticipant_delete_publisher(participant, publisher);
if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to delete publisher\n");
}

```

5.93 DataWriter Use Cases

Working with data writers.

Working with data writers.

5.93.1 Setting up a data writer

- Set up publisher (p. 981)
- Set up a topic (p. 978)
- Create a data writer, **FooDataWriter** (p. 2113), of user data type **Foo** (p. 2109) :

```
struct DDS_DataWriterQos writer_qos = DDS_DataWriterQos_INITIALIZER;
DDS_DataWriter* writer;
struct DDS_DataWriterListener writer_listener =
    DDS_DataWriterListener_INITIALIZER;
/* MyWriterListener_* functions are user defined to match
   DDS_DataWriterListener functions */
writer_listener.on_offered_deadline_missed =
    MyWriterListener_OfferedDeadlineMissed;
writer_listener.on_offered_incompatible_qos =
    MyWriterListener_OfferedIncompatibleQos;
writer_listener.on_liveliness_lost = MyWriterListener_LivelinessLost;
writer_listener.on_publication_match = MyWriterListener_PublicationMatch;
retcode = DDS_Publisher_get_default_datawriter_qos(publisher, &writer_qos);
if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to get default datawriter qos\n");
}
writer = DDS_Publisher_create_datawriter(publisher,
                                         topic,
                                         &writer_qos,
                                         &writer_listener /* or NULL */,
                                         DDS_STATUS_MASK_ALL);

if (writer == NULL) {
    printf("***Error: failed to create writer\n");
}
```

5.93.2 Managing instances

- Getting an instance "key" value of user data type **Foo** (p. 2109)


```
struct Foo* data = ...; /* user data */
retcode = FooDataWriter_get_key_value(writer, data, &instance_handle);
if (retcode != DDS_RETCODE_OK) {
    /* ... check for cause of failure */
}
```
- Registering an instance of type **Foo** (p. 2109)


```
DDS_InstanceHandle_t instance_handle = DDS_HANDLE_NIL;
instance_handle = FooDataWriter_register_instance(writer, data);
```
- Unregistering an instance of type **Foo** (p. 2109)


```
retcode = FooDataWriter_unregister_instance(writer, data, &instance_handle);
if (retcode != DDS_RETCODE_OK) {
    /* ... check for cause of failure */
}
```
- Disposing of an instance of type **Foo** (p. 2109)


```
retcode = FooDataWriter_dispose(writer, data, &instance_handle);
if (retcode != DDS_RETCODE_OK) {
    /* ... check for cause of failure */
}
```

5.93.3 Sending data

- Set up data writer (p. 982)
- Register instance (p. 982)
- Write instance of type **Foo** (p. 2109)


```
struct Foo* data = ...; /* user data */
DDS_InstanceHandle_t instance_handle =
    DDS_HANDLE_NIL; /* or a valid registered handle */
DDS_ReturnCode_t retcode;
retcode = FooDataWriter_write(writer, data, &instance_handle);
if (retcode != DDS_RETCODE_OK) {
    /* ... check for cause of failure */
}
```

5.93.4 Tearing down a data writer

- Delete DDS_DataWriter:


```
DDS_ReturnCode_t retcode;
retcode = DDS_Publisher_delete_datawriter(publisher, writer);
if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to delete writer\n");
}
```

5.94 Subscriber Use Cases

Working with subscribers.

Working with subscribers.

5.94.1 Setting up a subscriber

- Set up participant (p. 976)
- Create a DDS_Subscriber


```
struct DDS_SubscriberQos subscriber_qos = DDS_SubscriberQos_INITIALIZER;
DDS_Subscriber* subscriber;
struct DDS_SubscriberListener subscriber_listener = DDS_SubscriberListener_INITIALIZER;
DDS_ReturnCode_t retcode;
/* MySubscriberListener_* functions are user defined to match
   DDS_SubscriberListener functions */
subscriber_listener.on_data_on_readers =
    MySubscriberListener_DataOnReaders;
subscriber_listener.as_datareaderlistener.on_requested_deadline_missed =
    MySubscriberListener_RequestedDeadlineMissed;
subscriber_listener.as_datareaderlistener.on_requested_incompatible_qos =
    MySubscriberListener_RequestedIncompatibleQos;
subscriber_listener.as_datareaderlistener.on_sample_rejected =
    MySubscriberListener_SampleRejected;
subscriber_listener.as_datareaderlistener.on_liveliness_changed =
    MySubscriberListener_LivelinessChanged;
subscriber_listener.as_datareaderlistener.on_data_available =
    MySubscriberListener_DataAvailable;
subscriber_listener.as_datareaderlistener.on_subscription_matched =
    MySubscriberListener_SubscriptionMatch;
subscriber_listener.as_datareaderlistener.on_sample_lost =
    MySubscriberListener_SampleLost;
retcode = DDS_DomainParticipant_get_default_subscriber_qos(participant,
    &subscriber_qos);
if (retcode != DDS_RETCODE_OK) {
```

```

    printf("***Error: failed to get default subscriber qos\n");
}
subscriber = DDS_DomainParticipant_create_subscriber(participant,
                                                    &subscriber_qos,
                                                    &subscriber_listener /*or NULL*/,
                                                    DDS_STATUS_MASK_ALL);

if (subscriber == NULL) {
    printf("***Error: failed to create subscriber\n");
}

```

5.94.2 Set up subscriber to access received data

- **Set up subscriber** (p. 983)
- 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 `DDS_SubscriberListener` associated with the subscriber** (p. 989)
 - The processing to handle the status change is done in the `DDS_SubscriberListener_on_data_on_readers()` method of the attached listener.
 - Typical processing will **access the received data** (p. 984), either in arbitrary order or in a **coherent and ordered manner** (p. 985).
- **Enable `DDS_DATA_ON_READERS_STATUS` for the `DDS_StatusCondition` associated with the subscriber** (p. 990)
 - The processing to handle the status change is done **when the subscriber's attached status condition is triggered** (p. 991) and the `DDS_DATA_ON_READERS_STATUS` status on the subscriber is changed.
 - Typical processing will **access the received data** (p. 984), either in an arbitrary order or in a **coherent and ordered manner** (p. 985).

5.94.3 Access received data via a subscriber

- **Ensure subscriber is set up to access received data** (p. 984)
- **Get the list of readers that have data samples available:**

```

struct DDS_DataReaderSeq reader_seq = DDS_SEQUENCE_INITIALIZER;
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 = DDS_RETCODE_OK;
retcode = DDS_Subscriber_get_datareaders(subscriber,
                                         &reader_seq,
                                         max_samples,
                                         sample_state_mask,
                                         view_state_mask,
                                         instance_state_mask);

if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to access received data via subscriber\n");
    return;
}

```
- Upon successfully getting the list of readers with data, process the data readers to either:
 - **Read the data in each reader** (p. 988), **OR**
 - **Take the data in each reader** (p. 987)

If the intent is to access the data **coherently or in order** (p. 985), the list of data readers *must* be processed in the order returned:

```
int i;
for(i = 0; i < DDS_DataReaderSeq_get_length(&reader_seq); ++i) {

    TDataReader* reader = DDS_DataReaderSeq_get_reference(&reader_seq, i);
    /* Take the data from reader,
       OR
       Read the data from reader */
}
```

- **Alternatively**, call **DDS_Subscriber_notify_datareaders** (p.590) to invoke the **DDS_DataReaderListener** (p. 1593) for each of the data readers.

```
DDS_ReturnCode_t retcode;
retcode = DDS_Subscriber_notify_datareaders(subscriber);
if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to notify datareaders\n");
}
```

5.94.4 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. 1890) attached to a DDS_Subscriber:

- **Ensure subscriber is set up to access received data** (p. 984)
- Indicate that data will be accessed via the subscriber:


```
retcode = DDS_Subscriber_begin_access(subscriber);
if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to begin access\n");
}
```
- **Access received data via the subscriber, making sure that the data readers are processed in the order returned.** (p. 984)
- Indicate that the data access via the subscriber is done:


```
retcode = DDS_Subscriber_end_access(subscriber);
if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to end access\n");
}
```

5.94.5 Tearing down a subscriber

- **Delete DDS_Subscriber:**

```
DDS_ReturnCode_t retcode;
retcode = DDS_DomainParticipant_delete_subscriber(participant, subscriber);
if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to delete subscriber\n");
}
```

5.95 DataReader Use Cases

Working with data readers.

Working with data readers.

5.95.1 Setting up a data reader

- **Set up subscriber** (p. 983)
- **Set up a topic** (p. 978)
- **Create a data reader, `FooDataReader`** (p. 2112), of user data type **`Foo`** (p. 2109) :

```
struct DDS_DataReaderQos reader_qos = DDS_DataReaderQos_INITIALIZER;
DDS_DataReader* reader;
struct DDS_DataReaderListener reader_listener =
    DDS_DataReaderListener_INITIALIZER;
DDS_ReturnCode_t retcode;
/* MyReaderListener_* functions are user defined to match
   DDS_ReaderListener functions */
reader_listener.on_requested_deadline_missed =
    MyReaderListener_RequestedDeadlineMissed;
reader_listener.on_requested_incompatible_qos =
    MyReaderListener_RequestedIncompatibleQos;
reader_listener.on_sample_rejected = MyReaderListener_SampleRejected;
reader_listener.on_liveliness_changed =
    MyReaderListener_LivelinessChanged;
reader_listener.on_data_available = MyReaderListener_DataAvailable;
reader_listener.on_subscription_match =
    MyReaderListener_SubscriptionMatch;
reader_listener.on_sample_lost = MyReaderListener_SampleLost;
retcode = DDS_Subscriber_get_default_datareader_qos(
    subscriber, &reader_qos);
if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to get default datareader qos\n");
}
reader = DDS_Subscriber_create_datareader(subscriber,
    DDS_Topic_as_topicdescription(topic),
    &reader_qos,
    &reader_listener /* or NULL */,
    DDS_STATUS_MASK_ALL);
if (reader == NULL) {
    printf("***Error: failed to create reader\n");
}
```

5.95.2 Managing instances

- **Given a data reader**
`FooDataReader* reader = ...;`
- **Getting an instance "key" value of user data type `Foo`** (p. 2109)

```
struct Foo* data = ...; /* user data of type Foo */
retcode = FooDataReader_get_key_value(reader, data, &instance_handle);
if (retcode != DDS_RETCODE_OK) {
    /* ... check for cause of failure */
}
```

5.95.3 Set up reader to access received data

- **Set up data reader** (p. 986)
- 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 `DDS_DataReaderListener` associated with the data reader** (p. 989)
 - The processing to handle the status change is done in the `DDS_DataReaderListener_on_data_available()` method of the attached listener.
 - Typical processing will **access the received data** (p. 987).

- **Enable DDS_DATA_AVAILABLE_STATUS for the DDSStatusCondition associated with the data reader** (p. 990)
 - The processing to handle the status change is done **when the data reader's attached status condition is triggered** (p. 991) and the DDS_DATA_AVAILABLE_STATUS status on the data reader is changed.
 - Typical processing will **access the received data** (p. 987).

5.95.4 Access received data via a reader

- **Ensure reader is set up to access received data** (p. 986)
- Access the received data, by either:
 - **Taking the received data in the reader** (p. 987), **OR**
 - **Reading the received data in the reader** (p. 988)

5.95.5 Taking data

- **Ensure reader is set up to access received data** (p. 986)
- Take samples of user data type T. The samples are removed from the Service. The caller is responsible for deallocating the buffers.

```
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;
struct FooSeq data_seq = FooSeq_INITIALIZER;
struct DDS_SampleInfoSeq info_seq = DDS_SampleInfoSeq_INITIALIZER;
DDS_ReturnCode_t retcode;
retcode = FooDataReader_take(reader, &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) {
    printf("***Error: failed to access data from the reader\n");
}
```

- **Use the received data**

```
struct Foo* data = NULL;
struct DDS_SampleInfo* info = NULL;
int i;
for(i = 0; i < FooSeq_get_length(&data_seq); ++i) {
    info = DDS_SampleInfoSeq_get_reference(&info_seq, i);
    data = FooSeq_get_reference(&data_seq, i);
    /* Note that depending on the info->sample_state
       it is possible that data will be NULL
    */
    /* ... */
}
```

- **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.

```
DDS_ReturnCode_t retcode;
retcode = FooDataReader_return_loan(reader, &data_seq, &info_seq);
if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to return loan\n");
}
```

5.95.6 Reading data

- **Ensure reader is set up to access received data** (p. 986)
- Read samples of user data type **Foo** (p. 2109). The samples are not removed from the Service. It remains responsible for deallocating the buffers.

```
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;
struct FooSeq data_seq = FooSeq_INITIALIZER;
struct DDS_SampleInfoSeq info_seq = DDS_SampleInfoSeq_INITIALIZER;
DDS_ReturnCode_t retcode;
retcode = FooDataReader_read(reader, &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) {
    printf("***Error: failed to access data from the reader\n");
}
```

- **Use the received data**

```
struct Foo* data = NULL;
struct DDS_SampleInfo* info = NULL;
int i;
for(i = 0; i < FooSeq_get_length(&data_seq); ++i) {
    info = DDS_SampleInfoSeq_get_reference(&info_seq, i);
    data = FooSeq_get_reference(&data_seq, i);
    /* Note that depending on the info->sample_state
     * it is possible that data will be NULL
     */
    /* ... */
}
```

- **Return the data samples and the information buffers back to the middleware**

```
retcode = FooDataReader_return_loan(reader, &data_seq, &info_seq);
if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to return loan\n");
}
```

5.95.7 Tearing down a data reader

- **Delete DDS_DataReader:**

```
DDS_ReturnCode_t retcode;
retcode = DDS_Subscriber_delete_datareader(subscriber, reader);
if (retcode != DDS_RETCODE_OK) {
    printf("***Error: failed to delete reader\n");
}
```

5.96 Entity Use Cases

Working with entities.

Working with entities.

5.96.1 Enabling an entity

- To enable an **DDS_Entity** (p. 1360)

```
if (DDS_Entity_enable((DDS_Entity*)entity) != DDS_RETCODE_OK) {
    printf("***Error: failed to enable entity\n");
}
```

5.96.2 Checking if a status changed on an entity.

- Given an **DDS_Entity** (p. 1360) and a **DDS_StatusKind** (p. 1227) to check for, get the list of statuses that have changed since the last time they were respectively cleared.

```
DDS_StatusMask status_changes_mask;

status_changes_mask = DDS_Entity_get_status_changes(entity);
```

- 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. 1224) for details].

```
if (status_changes_mask & status_kind) {
    return 1; /* ... YES, status_kind changed ... */
} else {
    return 0; /* ... NO, status_kind did NOT change ... */
}
```

5.96.3 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. 1362)

```
if (DDS_DomainParticipant_get_qos(entity, &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. 1361).

```
switch (DDS_DomainParticipant_set_qos(entity, &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");
    }
}

DDS_DomainParticipantQos_finalize(&qos);
```

5.96.4 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 ... */
MyEntityListener_LivelinessChanged( /* one example method */
    void* listener_data,
    DDS_Entity* entity,
    const struct DDS_LivelinessChangedStatus *status);
```

- Set the entity_listener functions

```
/* ... set the listener struct to the previously defined functions ...
   one example is shown */
entity_listener.on_liveliness_changed = MyEntityListener_LivelinessChanged;
```

- Get an entity's listener using **get_listener (abstract)** (p. 1363)

```
entity_listener = DDS_Entity_get_listener(entity);
```

- 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. 1362). Only enable the listener for the statuses specified by the enabled_status_list.

```
if (DDS_Entity_set_listener(entity, &entity_listener, enabled_status_list)
    != DDS_RETCODE_OK) {
    printf("***Error: setting entity listener\n");
}
```

5.96.5 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 = DDS_Entity_get_statuscondition(entity);
```

- Get the list of statuses enabled for the status_condition

```
enabled_status_list = DDS_StatusCondition_get_enabled_statuses(statuscondition);
```

- 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 (DDS_StatusCondition_set_enabled_statuses(status_condition,
    enabled_status_list | status_kind)
    != DDS_RETCODE_OK) {
    /* ... check for cause of failure */
}
```

- Disable status_kind for the status_condition

```
if (DDS_StatusCondition_set_enabled_statuses(status_condition,
    enabled_status_list & ~status_kind)
    != DDS_RETCODE_OK) {
    /* ... check for cause of failure */
}
```

5.97 Waitset Use Cases

Using wait-sets and conditions.

Using wait-sets and conditions.

5.97.1 Setting up a wait-set

- Create a wait-set

```
DDS_WaitSet* waitset = DDS_WaitSet_new();
```

- Attach conditions

```
DDS_Condition* cond1 = ...;
DDS_Condition* cond2 = DDS_StatusCondition_as_condition(
    DDS_Entity_get_statuscondition(entity));
DDS_Condition* cond3 = DDS_ReadCondition_as_condition(
    DDS_DataReader_create_readcondition(
        reader,
        DDS_NOT_READ_SAMPLE_STATE,
        DDS_ANY_VIEW_STATE,
        DDS_ANY_INSTANCE_STATE));
DDS_Condition* cond4 = DDS_GuardCondition_as_condition(
    DDS_GuardCondition_new());
DDS_Condition* cond5 = ...;
DDS_ReturnCode_t retcode;
retcode = DDS_WaitSet_attach_condition(waitset, cond1);
if (retcode != DDS_RETCODE_OK) {
    /* ... error */
}
retcode = DDS_WaitSet_attach_condition(waitset, cond2);
if (retcode != DDS_RETCODE_OK) {
    /* ... error */
}
retcode = DDS_WaitSet_attach_condition(waitset, cond3);
if (retcode != DDS_RETCODE_OK) {
    /* ... error */
}
retcode = DDS_WaitSet_attach_condition(waitset, cond4);
if (retcode != DDS_RETCODE_OK) {
    /* ... error */
}
retcode = DDS_WaitSet_attach_condition(waitset, cond5);
if (retcode != DDS_RETCODE_OK) {
    /* ... error */
}
}
```

5.97.2 Waiting for condition(s) to trigger

- Set up a wait-set (p. 991)
- Wait for a condition to trigger or timeout, whichever occurs first

```
#define TRUE 1
#define FALSE 0
struct DDS_Duration_t timeout = { 0, 1000000 }; /* 1ms */
struct DDS_ConditionSeq active_conditions = DDS_SEQUENCE_INITIALIZER; /* holder for active conditions */
int is_cond1_triggered = FALSE;
int is_cond2_triggered = FALSE;
int i;
DDS_ReturnCode_t retcode;
retcode = DDS_WaitSet_wait(waitset, &active_conditions, &timeout);
if (retcode == DDS_RETCODE_TIMEOUT) {
    /* handle timeout */
} else if (retcode != DDS_RETCODE_OK) {
    /* check for other cause of failure */
} else {
    /* success */
    /* check if "cond1" or "cond2" are triggered: */
}
```

```

for(i = 0; i < DDS_ConditionSeq_get_length(&active_conditions); ++i) {
    if (DDS_ConditionSeq_get(&active_conditions, i) == cond1) {
        printf("Cond1 was triggered!");
        is_cond1_triggered = TRUE;
    }
    if (DDS_ConditionSeq_get(&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 ... */
}

```

5.97.3 Tearing down a wait-set

- Delete the wait-set

```

DDS_WaitSet_delete(waitset);
waitset = NULL;

```

5.98 Transport Use Cases

Working with pluggable transports.

Working with pluggable transports.

5.98.1 Changing the automatically registered built-in transports

- The **DDS_TRANSPORTBUILTIN_MASK_DEFAULT** (p. 1332) specifies the transport plugins that will be automatically registered with a newly created **DDS_DomainParticipant** (p. 77) by default.
- This default can be changed by changing the value of the value of **TRANSPORT_BUILTIN** (p. 1331) Qos Policy on the **DDS_DomainParticipant** (p. 77)
- To change the **DDS_DomainParticipantQos::transport_builtin** (p. 1729) Qos Policy:

```

struct DDS_DomainParticipantQos participant_qos = DDS_DomainParticipantQos_INITIALIZER;

DDS_DomainParticipantFactory_get_default_participant_qos(factory, &participant_qos);

participant_qos.transport_builtin.mask = DDS_TRANSPORTBUILTIN_SHMEM |
                                        DDS_TRANSPORTBUILTIN_UDPv4;

```


5.98.2 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 **DDS_DomainParticipantFactory** (p. 32) to create the participants disabled, as described in **Turning off auto-enable of newly created participant(s)** (p. 976)

- Get the property of the desired builtin transport plugin, say **::UDPV4 Transport** (p. 1041)

```
struct NDDS_Transport_UDPv4_Property_t property = NDDS_TRANSPORT_UDPv4_PROPERTY_DEFAULT;

if (NDDS_Transport_Support_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. 1041) 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. 1935) 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. 1041)

```
if (NDDS_Transport_Support_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. 988) to turn on communications with other participants in the domain using the new properties for the automatically registered builtin transport plugins.

5.98.3 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. 1041)

```
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");
}
```

5.98.4 Deleting a transport

- A transport plugin can only be deleted only after the **DDS_DomainParticipant** (p. 77) 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);
```

5.98.5 Registering a transport with a participant

The basic steps for setting up transport plugins for use in an RTI Connex application are described below.

- Tell the **DDS_DomainParticipantFactory** (p. 32) to create the participants disabled, as described in **Turning off auto-enable of newly created participant(s)** (p. 976)

Optionally Changing the automatically registered built-in transports (p. 992)

Optionally Changing the properties of the automatically registered builtin transports (p. 993)

- Create a disabled **DDS_DomainParticipant** (p. 77), as described in **Setting up a participant** (p. 976)
- Decide on the **network address** (p. 724) 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. 721) for the transport plugin. An alias can refer to one or more transport plugins. The transport class name (see **Builtin Transport Class Names** (p. 1358)) 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 */
struct DDS_StringSeq aliases = DDS_SEQUENCE_INITIALIZER;
if (!DDS_StringSeq_from_array(&aliases, ALIASES, ALIASES_LENGTH)) {
    printf("***Error: creating initializing aliases\n");
}
```

- Register the transport plugin with the **DDS_DomainParticipant** (p. 77). 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 = NDDS_Transport_Support_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");
}

/* Finalize the aliases StringSeq */
DDS_StringSeq_finalize(&aliases);
```

Optionally Adding receive routes for a transport (p. 995)

Optionally Adding send routes for a transport (p. 995)

- **Enable the participant** (p. 988) to turn on communications with other participants in the domain, using the newly registered transport plugins, and automatically registered builtin transport plugins (if any).

5.98.6 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 (NDDS_Transport_Support_add_receive_route(&handle, &subnet, 112)
    != DDS_RETCODE_OK) {
    printf("***Error: adding receive route\n");
}
```

5.98.7 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 (NDDS_Transport_Support_add_send_route(&handle, &subnet, 120)
    != DDS_RETCODE_OK) {
    printf("***Error: adding send route\n");
}
```

5.99 Filter Use Cases

Working with data filters.

Working with data filters.

5.99.1 Introduction

RTI Connex supports filtering data either during the exchange from **DDS_DataWriter** (p. 485) to **DDS_DataReader** (p. 615), or after the data has been stored at the **DDS_DataReader** (p. 615).

Filtering during the exchange process is performed by a **DDS_ContentFilteredTopic** (p. 177), which is created by the **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615) is performed by creating a **DDS_QueryCondition** (p. 699), which can then be 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. 630) or **FooDataReader_take_w_condition** (p. 632) functions. (Conditions may also be used with the APIs **FooDataReader_read_next_instance_w_condition** (p. 643) and **FooDataReader_take_next_instance_w_condition** (p. 645).)

Filtering may be performed on any topic, either keyed or un-keyed, except the **Built-in Topics** (p. 165). Filtering may be performed 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. 313) API.

RTI Connex contains built-in support for filtering using SQL syntax, described in the **Queries and Filters Syntax** (p. 738) module.

5.99.1.1 Overview of ContentFilteredTopic

Each **DDS_ContentFilteredTopic** (p.177) is created based on an existing **DDS_Topic** (p.175). The **DDS_Topic** (p.175) specifies the **field_names** and **field_types** of the data contained within the topic. The **DDS_ContentFilteredTopic** (p.177), by means of its **filter_expression** and **expression_parameters**, further specifies the *values* of the data which the **DDS_DataReader** (p.615) wishes to receive.

Custom filters may also be constructed and utilized as described in the **Creating Custom Content Filters** (p.998) module.

Once the **DDS_ContentFilteredTopic** (p.177) has been created, a **DDS_DataReader** (p.615) can be created using the filtered topic. The filter's characteristics are exchanged between the **DDS_DataReader** (p.615) and any matching **DDS_DataWriter** (p.485) during the discovery process.

If the **DDS_DataWriter** (p.485) allows (by **DDS_DataWriterResourceLimitsQosPolicy::max_remote_reader_filters** (p.1676)), and the **DDS_DataReader** (p.615) 's **DDS_TransportMulticastQosPolicy** (p.2058) is empty, then the **DDS_DataWriter** (p.485) will filter out samples that don't meet the filtering criteria.

If disallowed by the **DDS_DataWriter** (p.485), or the **DDS_DataReader** (p.615) has set the **DDS_TransportMulticastQosPolicy** (p.2058), then the **DDS_DataWriter** (p.485) sends all samples to the **DDS_DataReader** (p.615), and the **DDS_DataReader** (p.615) discards any samples that do not meet the filtering criteria.

The **expression_parameters** can be modified using **DDS_ContentFilteredTopic_set_expression_parameters** (p.202). Any changes made to the filtering criteria by means of **DDS_ContentFilteredTopic_set_expression_parameters** (p.202), will be conveyed to any connected **DDS_DataWriter** (p.485). New samples will be subject to the modified filtering criteria, but samples that have already been accepted or rejected are unaffected. However, if the **DDS_DataReader** (p.615) connects to a **DDS_DataWriter** (p.485) that *re-sends* its data, the re-sent samples will be subjected to the new filtering criteria.

5.99.1.2 Overview of QueryCondition

DDS_QueryCondition (p.699) combine aspects of the content filtering capabilities of **DDS_ContentFilteredTopic** (p.177) with state filtering capabilities of **DDS_ReadCondition** (p.694) to create a reconfigurable means of filtering or searching data in the **DDS_DataReader** (p.615) queue.

DDS_QueryCondition (p.699) may be created on a disabled **DDS_DataReader** (p.615), or after the **DDS_DataReader** (p.615) has been enabled. If the **DDS_DataReader** (p.615) is enabled, and has already received and stored samples in its queue, then all data samples in the are filtered against the **DDS_QueryCondition** (p.699) filter criteria at the time that the **DDS_QueryCondition** (p.699) is created. (Note that an exclusive lock is held on the **DDS_DataReader** (p.615) sample queue for the duration of the **DDS_QueryCondition** (p.699) creation).

Once created, incoming samples are filtered against all **DDS_QueryCondition** (p.699) filter criteria at the time of their arrival and storage into the **DDS_DataReader** (p.615) queue.

The number of **DDS_QueryCondition** (p.699) filters that an individual **DDS_DataReader** (p.615) may create is set by **DDS_DataReaderResourceLimitsQosPolicy::max_query_condition_filters** (p.1632), to an upper maximum of 32.

5.99.2 Filtering with ContentFilteredTopic

- Set up subscriber (p. 983)
- Set up a topic (p. 978)
- Create a ContentFilteredTopic, of user data type **Foo** (p. 2109) :

```
DDS_ContentFilteredTopic *cft = NULL;
struct DDS_StringSeq cft_parameters;
const char* cft_param_list[] = {"1", "100"};
DDS_StringSeq_initialize(&cft_parameters);
DDS_StringSeq_set_maximum(&cft_parameters, 2);
DDS_StringSeq_from_array(&cft_parameters, param_list, 2);
cft = DDS_DomainParticipant_create_contentfilteredtopic(participant,
    "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:

```
DDS_DataReader* reader;
FooDataReader* Foo_reader;
reader = DDS_Subscriber_create_datareader(subscriber,
    DDS_ContentFilteredTopic_as_topicdescription(cft),
    &reader_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;
}
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 **DDS_ContentFilteredTopic** (p. 177) is exactly the same as normal reads or takes, as described in **DataReader Use Cases** (p. 985).

- Changing filter criteria using set_expression_parameters:

```
DDS_String_free(DDS_StringSeq_get(&parameters, 0));
DDS_String_free(DDS_StringSeq_get(&parameters, 1));
*DDS_StringSeq_get_reference(&parameters, 0) = DDS_String_dup("5");
*DDS_StringSeq_get_reference(&parameters, 1) = DDS_String_dup("9");
retcode = DDS_ContentFilteredTopic_set_expression_parameters(cft,
    &parameters);

if (retcode != DDS_RETCODE_OK) {
    printf("set_expression_parameters error\n");
    subscriber_shutdown(participant);
    return -1;
}
```

5.99.3 Filtering with Query Conditions

- Given a data reader of type **Foo** (p. 2109)


```
DDS_DataReader* reader = ...;
FooDataReader* Foo_reader = FooDataReader_narrow(reader);
```
- Creating a QueryCondition


```
DDS_QueryCondition *queryCondition = NULL;
struct DDS_StringSeq qc_parameters;
const char *qc_param_list[] = {"1", "100"};
DDS_StringSeq_initialize(&qc_parameters);
```

```

DDS_StringSeq_set_maximum(&qc_parameters, 2);
DDS_StringSeq_from_array(&qc_parameters, qc_param_list, 2);
queryCondition = DDS_DataReader_create_querycondition(reader,
                                                    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");
    subscriber_shutdown(participant);
    return -1;
}

```

- **Reading matching samples with a DDS_QueryCondition** (p. 699)

```

struct FooSeq data_seq = DDS_SEQUENCE_INITIALIZER;
struct DDS_SampleInfoSeq info_seq = DDS_SEQUENCE_INITIALIZER;
retcode = FooDataReader_read_w_condition(Foo_reader,
                                        &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 < FooSeq_get_length(&data_seq); ++i) {
        if (DDS_SampleInfoSeq_get_reference(&info_seq, i)->valid_data) {
            /* process your data here */
        }
    }
    retcode = FooDataReader_return_loan(Foo_reader,
                                       &data_seq, &info_seq);

    if (retcode != DDS_RETCODE_OK) {
        printf("return loan error %d\n", retcode);
        subscriber_shutdown(participant);
        return -1;
    }
}

```

- **DDS_QueryCondition_set_query_parameters** (p. 700) is used similarly to **DDS_ContentFilteredTopic_set_expression_parameters** (p. 202), and the same coding techniques can be used.
- Any **DDS_QueryCondition** (p. 699) that have been created must be deleted before the **DDS_DataReader** (p. 615) can be deleted. This can be done using **DDS_DataReader_delete_contained_entities** (p. 674) or manually as in:

```

retcode = DDS_DataReader_delete_readcondition(reader,
                                              queryCondition);

```

5.99.4 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 **DDS_DataReader** (p. 615) caches the results of the filter (pass or not pass) for each instance. When another sample of the same instance is seen at the **DDS_DataReader** (p. 615), 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 **DDS_DataReader** (p. 615), for either **DDS_ContentFilteredTopic** (p. 177) or **DDS_QueryCondition** (p. 699). This does *not* apply to filtering performed for **DDS_ContentFilteredTopic** (p. 177) by the **DDS_DataWriter** (p. 485).

5.100 Creating Custom Content Filters

Working with custom content filters.

Working with custom content filters.

5.100.1 Introduction

By default, RTI Connexx 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 than SQL

This HOWTO explains how to write your own custom filter and is divided into the following sections:

- **The Custom Content Filter API** (p. 999)
- **Example Using C format strings** (p. 1000)

5.100.2 The Custom Content Filter API

A custom content filter is created by calling the `DDS_DomainParticipant_register_contentfilter` (p. 120) function with a `DDS_ContentFilter` (p. 1571) that contains a `compile`, an `evaluate` function and a `finalize` function. `DDS_ContentFilteredTopic` (p. 177) can be created with `DDS_DomainParticipant_create_contentfilteredtopic_with_filter` (p. 119) to use this filter.

A custom content filter is used by RTI Connexx at the following times during the life-time of a `DDS_ContentFilteredTopic` (p. 177) (the function called is shown in parenthesis).

- When a `DDS_ContentFilteredTopic` (p. 177) is created (`compile` (p. 999))
- When the filter parameters are changed on the `DDS_ContentFilteredTopic` (p. 177) (`compile` (p. 999)) with `DDS_ContentFilteredTopic_set_expression_parameters` (p. 202)
- When a sample is filtered (`evaluate` (p. 999)). This function is called by the RTI Connexx core with a de-serialized sample
- When a `DDS_ContentFilteredTopic` (p. 177) is deleted (`finalize` (p. 1000))

5.100.2.1 The compile function

The `compile` (p. 1000) 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 `DDS_ContentFilter::compile` (p. 1572) for details.

5.100.2.2 The evaluate function

The `evaluate` (p. 1000) function is called each time a sample is received to determine if a sample should be filtered out and discarded.

See `DDS_ContentFilter::evaluate` (p. 1574) for details.

5.100.2.3 The finalize function

The **finalize** (p. 1001) 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 **DDS_ContentFilter::finalize** (p. 1576) for details.

5.100.3 Example Using C format strings

Assume that you have a type **Foo** (p. 2109).

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. 2109).

5.100.3.1 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. 999) function.

```
DDS_ReturnCode_t
howto_write_simple_compile_function(void *handle,
                                   void **new_compile_data,
                                   const char *expression,
                                   const struct DDS_StringSeq *parameters,
                                   const struct DDS_TypeCode *type_code,
                                   const char *type_class_name,
                                   void *old_compile_data)
{
    *new_compile_data = (void*)DDS_String_dup(*DDS_StringSeq_get_reference(parameters,0));
    return DDS_RETCODE_OK;
}
```

5.100.3.2 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. 999) function.

```
DDS_Boolean
howto_write_simple_evaluate_function(void *filter_data,
                                   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);
}
```


5.100.3.3 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. 1000) function.

```
void
howto_write_simple_finalize_function(void *filter_data,
                                   void *compile_data)
{
    /* free parameter string from compile function */
    DDS_String_free((char *)compile_data);
}
```

5.100.3.4 Registering the Filter

Before the custom filter can be used, it must be registered with RTI Connex:

```
struct DDS_ContentFilter filter = DDS_ContentFilter_INITIALIZER;
filter.compile = howto_write_simple_compile_function;
filter.evaluate = howto_write_simple_evaluate_function;
filter.finalize = howto_write_simple_finalize_function;
filter.filter_data = NULL;
if (DDS_DomainParticipant_register_contentfilter(
    participant, "MyCustomFilter", &filter) != DDS_RETCODE_OK) {
    printf("Failed to register custom filter\n");
}
```

5.100.3.5 Unregistering the Filter

When the filter is no longer needed, it can be unregistered from RTI Connex:

```
if (DDS_DomainParticipant_unregister_contentfilter(
    participant, "MyCustomFilter" ) != DDS_RETCODE_OK) {
    printf("Failed to unregister custom filter\n");
}
```

5.101 Large Data Use Cases

Working with large data types.

Working with large data types.

5.101.1 Introduction

RTI Connex supports data types whose size exceeds the maximum message size of the underlying transports. A **DDS_DataWriter** (p. 485) 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 **DDS_DataReader** (p. 615) 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. 1951) and **DDS_KEEP_LAST_HISTORY_QOS** (p. 1293).

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.

5.101.2 Writing Large Data

In order to use the large data feature with the **DDS_RELIABLE_RELIABILITY_QOS** (p. 1323) setting, the **DDS_↵DataWriter** (p. 485) must be configured as an asynchronous writer (**DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS** (p. 1319)) with associated **DDS_FlowController** (p. 559).

While the use of an asynchronous writer and flow controller is optional when using the **DDS_BEST_EFFORT_↵RELIABILITY_QOS** (p. 1323) 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. 982)
- **Add flow control** (p. 979)

5.101.3 Receiving Large Data

Large data is supported by default and in most cases, no further changes are required.

The **DDS_DataReaderResourceLimitsQosPolicy** (p. 1619) allows tuning the resources available to the **DDS_Data_↵Reader** (p. 615) for reassembling fragmented large data.

- **Set up reader** (p. 986)

5.102 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.

5.102.1 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 Connex 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. 305),
- **Using DynamicData** (p. 313)

Set up

- **Create a DomainParticipant** (p. 976)
- **Create a requester** (p. 1004)
- **Create a requester with parameters** (p. 1004)
- **Create a replier** (p. 1007)

Requester: sending requests and receiving replies

- **Requester example** (p. 1005)
- **Taking loaned samples** (p. 1005)
- **Correlation between requests and replies** (p. 1006)

Replier: receiving requests and sending replies

- **Replier example** (p. 1008)
- **SimpleReplier example** (p. 1009)
- **Taking loaned samples** (p. 1005)

Note

To use Request-Reply you need to build and link your application with the additional `rticonnextmsgc` library.

The examples in this section require the following header files:

```
#include "connext_c/connext_c_requester.h"
#include "connext_c/connext_c_replier.h"
#include "connext_c/connext_c_simple_replier.h"
```

5.102.2 Creating a Requester

- **Setting up a participant** (p. 976)
- Instantiate the declaration of a **FooBarRequester** (p. 2111), a requester that sends requests of type **Foo** (p. 2109) and receives replies of type **Bar** (for example, in FooBarRequester.h)

```
/* Instantiate the FooBarRequester structure and declare its functions, for
 * example in a file named FooBarRequester.h
 *
 * The request type is Foo
 * The reply type is Bar
 * The name of the requester struct and prefix for the operations is FooBarRequester
 *
 */
RTI_Connext_REQUESTER_DECL(Foo, Bar, FooBarRequester)
```

- Instantiate the implementation of a **FooBarRequester** (p. 2111) (for example, in FooBarRequester.c)

```
/* Instantiate the implementation of the FooBarRequester functions,
 * for example in a file named FooBarRequester.c.
 *
 *
 */
#define TReq Foo
#define TRep Bar
#define TRequester FooBarRequester
#include "connext_c/generic/connext_c_requestreply_TReqTRepRequester.gen"
```

- **Create a Requester**

```
DDS_DomainParticipant * participant;
FooBarRequester * requester;
/* Create a DomainParticipant */
participant = DDS_DomainParticipantFactory_create_participant(
    DDS_DomainParticipantFactory_get_instance(),
    domain_id, &DDS_PARTICIPANT_QOS_DEFAULT,
    NULL, DDS_STATUS_MASK_NONE);
if (participant == NULL) {
    printf("Error creating participant\n");
}
/* Create a Requester */
requester = FooBarRequester_create(participant, "TestService");
if (requester == NULL) {
    printf("Error creating requester\n");
}
```

5.102.3 Creating a Requester with optional parameters

- **Setting up a participant** (p. 976)
- **Creating a Requester with optional parameters**

```
DDS_DomainParticipant * participant;
FooBarRequester * requester;
struct RTI_Connext_RequesterParams params =
    RTI_Connext_RequesterParams_INITIALIZER;
/* Create a DomainParticipant */
participant = DDS_DomainParticipantFactory_create_participant(
    DDS_DomainParticipantFactory_get_instance(),
    domain_id, &DDS_PARTICIPANT_QOS_DEFAULT,
    NULL, DDS_STATUS_MASK_NONE);
if (participant == NULL) {
    printf("Error creating participant\n");
}
/* Set the parameters */
/* The participant is always required */
params.participant = participant;
/* The service name is required, or alternatively
   params.request_topic_name and params.reply_topic_name can be used */
params.service_name = (char *) "TestService";
/* Create a Requester with a QoS profile (located for example in
   USER_QOS_PROFILES.xml, in the current working directory) */
params.qos_library_name = (char *) "RequestReplyExampleProfiles";
params.qos_profile_name = (char *) "RequesterExampleProfile";
/* Create a Requester */
```

```

requester = FooBarRequester_create_w_params(&params);
if (requester == NULL) {
    printf("Error creating requester\n");
}

```

See also

Creating a Requester (p. 1004)

Configuring Request-Reply QoS profiles (p. 1010)

5.102.4 Requester example

- **Creating a Requester** (p. 1004)
- **Creating a Requester with optional parameters** (p. 1004)

- **Requester example**

```

DDS_ReturnCode_t retcode;
Foo * request;
Bar * reply;
struct DDS_SampleInfo info = DDS_SAMPLEINFO_DEFAULT;
/* Note: error checking omitted for simplicity */
/* Send request */
request = FooTypeSupport_create_data();
strcpy(request->message, "A Request");
retcode = FooBarRequester_send_request(requester, request);
/* Receive reply (wait for it and get the sample) */
reply = BarTypeSupport_create_data();
retcode = FooBarRequester_receive_reply(requester, reply, &info, &MAX_WAIT);
if (retcode == DDS_RETCODE_OK) {
    if(info.valid_data) {
        printf("Received reply: %s\n", reply->message);
    } else {
        printf("Received invalid reply\n");
    }
} else if (retcode == DDS_RETCODE_TIMEOUT || retcode == DDS_RETCODE_NO_DATA) {
    printf("Reply not received\n");
} else {
    printf("Error receiving reply\n");
}
/* Delete the samples */
FooTypeSupport_delete_data(request);
BarTypeSupport_delete_data(reply);

```

See also

Replier example (p. 1008)

SimpleReplier example (p. 1009)

5.102.5 Taking loaned samples

- **Creating a Requester** (p. 1004)
- **Get iterator to loaned replies (no copies)**

```

int i;
DDS_ReturnCode_t retcode;
struct BarSeq replies = DDS_SEQUENCE_INITIALIZER;
struct DDS_SampleInfoSeq infos = DDS_SEQUENCE_INITIALIZER;
/* 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)
*/
retcode = FooBarRequester_take_replies(

```

```

    requester, &replies, &infos, DDS_LENGTH_UNLIMITED);
if (retcode == DDS_RETCODE_NO_DATA) {
    printf("No replies are available\n");
    return;
} else if (retcode != DDS_RETCODE_OK) {
    printf("Error getting replies\n");
    return;
}
for (i = 0; i < BarSeq_get_length(&replies); i++) {
    Bar * reply = BarSeq_get_reference(&replies, i);
    struct DDS_SampleInfo * info = DDS_SampleInfoSeq_get_reference(&infos, i);
    if (info->valid_data) {
        printf("Received reply: %s\n", reply->message);
    }
}
/* Return the loan */
FooBarRequester_return_loan(requester, &replies, &infos);

```

See also

Requester example (p. 1005)

Replier example (p. 1008)

5.102.6 Correlating requests and replies

- **Creating a Requester** (p. 1004)
- **Example 1) Waiting for a reply for a specific request**

```

DDS_ReturnCode_t retcode;
Foo * request;
struct DDS_WriteParams_t request_info1 = DDS_WRITEPARAMS_DEFAULT;
struct DDS_WriteParams_t request_info2 = DDS_WRITEPARAMS_DEFAULT;
Bar * reply;
struct DDS_SampleInfo reply_info;
struct DDS_SampleIdentity_t related_sample_id;
/* Note: error checking omitted for simplicity */
request = FooTypeSupport_create_data();
/* Send first request */
strcpy(request->message, "Request 1");
FooBarRequester_send_request_w_params(requester, request, &request_info1);
/* Send second request */
strcpy(request->message, "Request 2");
FooBarRequester_send_request_w_params(requester, request, &request_info2);
/* Wait for one reply to the second request */
retcode = RTI_Connext_Requester_wait_for_replies_for_related_request(
    (RTI_Connext_Requester *) requester, 1, &MAX_WAIT,
    &request_info2.identity);
if (retcode != DDS_RETCODE_OK) {
    printf("Did not receive reply for request 2\n");
    return;
}
/* Take that reply */
reply = BarTypeSupport_create_data();
retcode = FooBarRequester_take_reply_for_related_request(
    requester, reply, &reply_info, &request_info2.identity);
/* This postcondition should always be true */
DDS_SampleInfo_get_related_sample_identity(&reply_info, &related_sample_id);
if (!DDS_SampleIdentity_equals(&related_sample_id, &request_info2.identity)) {
    printf("postcondition failed\n");
}
if (reply_info.valid_data) {
    printf("Received reply for request 2: %s\n", reply->message);
}
/* Wait for one reply to the second request */
retcode = RTI_Connext_Requester_wait_for_replies_for_related_request(
    (RTI_Connext_Requester *) requester, 1, &MAX_WAIT,
    &request_info1.identity);
if (retcode != DDS_RETCODE_OK) {
    printf("Did not receive reply for request 1\n");
    return;
}

```

```

/* Take that reply */
retcode = FooBarRequester_take_reply_for_related_request(
    requester, reply, &reply_info, &request_info1.identity);
/* This postcondition should always be true */
DDS_SampleInfo_get_related_sample_identity(&reply_info, &related_sample_id);
if (!DDS_SampleIdentity_equals(&related_sample_id, &request_info1.identity)) {
    printf("postcondition failed\n");
}
if (reply_info.valid_data) {
    printf("Received reply for request 1: %s\n", reply->message);
}
FooTypeSupport_delete_data(request);
BarTypeSupport_delete_data(reply);

```

- Example 2) Correlate reply after receiving it

```

int i;
DDS_ReturnCode_t retcode;
Foo * request;
struct DDS_WriteParams_t request_info1 = DDS_WRITEPARAMS_DEFAULT;
struct DDS_WriteParams_t request_info2 = DDS_WRITEPARAMS_DEFAULT;
struct BarSeq replies = DDS_SEQUENCE_INITIALIZER;
struct DDS_SampleInfoSeq infos = DDS_SEQUENCE_INITIALIZER;
struct DDS_SampleIdentity_t related_sample_id;
/* Note: error checking omitted for simplicity */
request = FooTypeSupport_create_data();
/* Send first request */
strcpy(request->message, "Request 1");
FooBarRequester_send_request_w_params(requester, request, &request_info1);
/* Send second request */
strcpy(request->message, "Request 2");
FooBarRequester_send_request_w_params(requester, request, &request_info2);
/* Wait for two replies. In this case we don't mind the reception order */
retcode = RTI_Connext_Requester_wait_for_replies(
    (RTI_Connext_Requester *) requester, 2, &MAX_WAIT);
if (retcode != DDS_RETCODE_OK) {
    printf("Replies not received\n");
    return;
}
retcode = FooBarRequester_take_replies(
    requester, &replies, &infos, DDS_LENGTH_UNLIMITED);
for (i = 0; i < BarSeq_get_length(&replies); i++) {
    Bar * reply = BarSeq_get_reference(&replies, i);
    struct DDS_SampleInfo * info = DDS_SampleInfoSeq_get_reference(&infos, i);
    DDS_SampleInfo_get_related_sample_identity(info, &related_sample_id);
    if (DDS_SampleIdentity_equals(&related_sample_id, &request_info1.identity)) {
        printf("Received reply for request 1: %s\n", reply->message);
    } else if (DDS_SampleIdentity_equals(&related_sample_id, &request_info2.identity)) {
        printf("Received reply for request 2: %s\n", reply->message);
    } else {
        /* Should not happen */
        printf("Received unexpected reply\n");
    }
}
/* Return the loan */
FooBarRequester_return_loan(requester, &replies, &infos);
FooTypeSupport_delete_data(request);

```

See also

Requester example (p. 1005)

Replier example (p. 1008)

5.102.7 Creating a Replier

- **Setting up a participant** (p. 976)
- Instantiate the declaration of a **FooBarReplier** (p. 2110), a repplier that receives requests of type **Foo** (p. 2109) and sends replies of type **Bar** (for example, in **FooBarReplier.h**)

```

/* Instantiate the FooBarReplier structure and declare its functions, for
 * example in a file named FooBarReplier.h
 */

```

```

    * The request type is Foo
    * The reply type is Bar
    * The name of the replier struct and prefix for the operations is FooBarReplier
    */
RTI_CONNEXT_REPLIER_DECL(Foo, Bar, FooBarReplier)

```

- Instantiate the implementation of a **FooBarReplier** (p. 2110) (for example, in FooBarReplier.c)

```

/* Instantiate the implementation of the FooBarReplier functions,
 * for example in a file named FooBarReplier.c.
 *
 *
 */
#define TReq Foo
#define TRep Bar
#define TReplier FooBarReplier
#include "connext_c/generic/connext_c_requestreply_TReqTRepReplier.gen"

```

- Create a Replier

```

DDS_DomainParticipant * participant;
FooBarReplier * replier;
/* Create a DomainParticipant */
participant = DDS_DomainParticipantFactory_create_participant(
    DDS_DomainParticipantFactory_get_instance(),
    domain_id, &DDS_PARTICIPANT_QOS_DEFAULT,
    NULL, DDS_STATUS_MASK_NONE);
if (participant == NULL) {
    printf("Error creating participant\n");
}
/* Create a Replier */
replier = FooBarReplier_create(participant, "TestService");
if (replier == NULL) {
    printf("Error creating replier\n");
}

```

5.102.8 Replier example

- Creating a Replier (p. 1007)

- Replier example

```

DDS_ReturnCode_t retcode;
Foo * request;
Bar * reply;
struct DDS_SampleInfo request_info;
struct DDS_SampleIdentity_t request_id;
/* Note: error checking omitted for simplicity */
/* Receive one request */
request = FooTypeSupport_create_data();
retcode = FooBarReplier_receive_request(
    replier, request, &request_info, &MAX_WAIT);
if (retcode != DDS_RETCODE_OK) {
    printf("Request not received\n");
    FooTypeSupport_delete_data(request);
    return 0;
}
if (request_info.valid_data) {
    reply = BarTypeSupport_create_data();
    sprintf(reply->message, "Reply for %s", request->message);
    /* Send a reply for that request */
    DDS_SampleInfo_get_sample_identity(&request_info, &request_id);
    retcode = FooBarReplier_send_reply(replier, reply, &request_id);
    /* Note: a replier can send more than one reply for the same request */
    BarTypeSupport_delete_data(reply);
}
FooTypeSupport_delete_data(request);

```

See also

Requester example (p. 1005)

5.102.9 SimpleReplier example

- Instantiate the declaration of a **FooBarSimpleReplier** (p.2112), a replier that receives requests of type **Foo** (p.2109) and sends replies of type **Bar** (for example, in `FooBarSimpleReplier.h`)

```
/* Instantiate the FooBarSimpleReplier structure and declare its functions, for
 * example in a file named FooBarSimpleReplier.h
 *
 * The request type is Foo
 * The reply type is Bar
 * The name of the simple replier struct and prefix for the operations is
 * FooBarSimpleReplier
 *
 */
RTI_Connext_SIMPLEREPLIER_DECL(Foo, Bar, FooBarSimpleReplier)
```

- Instantiate the implementation of a **FooBarSimpleReplier** (p.2112) (for example, in `FooBarSimpleReplier.c`)

```
/* Instantiate the implementation of the FooBarSimpleReplier functions,
 * for example in a file named FooBarSimpleReplier.c.
 *
 *
 */
#define TReq Foo
#define TRep Bar
#define TSimpleReplier FooBarSimpleReplier
#include "connext_c/generic/connext_c_requestreply_TReqTRepSimpleReplier.gen"
```

- A **FooBarSimpleReplier** (p.2112) wraps a **FooBarReplier** (p.2110), which has to be defined as well. If you haven't already, instantiate the declaration and definition of **FooBarReplier** (p.2110) as described in **Creating a Replier** (p.1007) (you can do this in the same `.h` and `.c` files you just used for **FooBarSimpleReplier** (p.2112)).

- Implement a listener

```
void * MySimpleReplierListener_on_request_available(
    RTI_Connext_SimpleReplierListener * listener,
    const void * request_untyped, const struct DDS_SampleInfo * info)
{
    Bar * reply;
    Foo * request = (Foo *) request_untyped;
    if (!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;
    }
    reply = BarTypeSupport_create_data();
    sprintf(reply->message, "Simple reply for %s", request->message);
    return (void *) reply;
}

void MySimpleReplierListener_on_return_loan(
    RTI_Connext_SimpleReplierListener * listener,
    void * reply)
{
    /* Delete the data we created on request available
     *
     * Note: we can store a single reply sample in listener->user_data
     *       and reuse it to avoid reserving and deleting memory on every call
     *       to on_request_available/on_return_loan
     */
    BarTypeSupport_delete_data(reply);
}
```

- And create a SimpleReplier with the listener:

```
RTI_Connext_SimpleReplierListener listener;
FooBarSimpleReplier * replier;
/* Note: error checking omitted for simplicity */
listener.on_request_available = MySimpleReplierListener_on_request_available;
listener.return_loan = MySimpleReplierListener_on_return_loan;
replier = FooBarSimpleReplier_create(participant, "TestService", &listener);
/* After creation the SimpleReplier is already active and may call
 * the listener */
```

See also

Requester example (p.1005)

5.102.10 Configuring Request-Reply QoS profiles

If you do not specify your own quality of service parameters (in **RTI_Connext_RequesterParams** (p. 2173) and **RTI_Connext_ReplierParams** (p. 2169)), a **FooBarRequester** (p. 2111) and **FooBarReplier** (p. 2110) are created using a default configuration defined in the built-in profile **DDS_PROFILE_PATTERN_RPC** (p. 1412). You use this built-in profile as the base for your own profiles:

```
<?xml version="1.0"?>
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://community.rti.com/schema/current/rti_dds_profiles.xsd">
  <qos_library name="RequestReplyExampleProfiles">
    <!-- Use these profiles to customize the Requester or Replier QoS -->
    <!-- These profiles inherit from the built-in profile
        BuiltinQoSLib::Pattern.RPC, which defines the default values for
        the DataWriters and DataReaders created by a Requester or Replier
    -->
    <qos_profile name="RequesterExampleProfile" base_name="BuiltinQoSLib::Pattern.RPC">
      <!-- Set the QoS for the DataReader created by the Requester -->
      <!--
        <datareader_qos>
          ...
        </datareader_qos>
      -->
      <!-- Set the QoS for the DataWriter created by the Requester -->
      <!--
        <datawriter_qos>
          ...
        </datawriter_qos>
      -->
    </qos_profile>
    <qos_profile name="ReplierExampleProfile" base_name="BuiltinQoSLib::Pattern.RPC">
      <!-- Set the QoS for the DataReader created by the Replier -->
      <!--
        <datareader_qos>
          ...
        </datareader_qos>
      -->
      <!-- Set the QoS for the DataWriter created by the Replier -->
      <!--
        <datawriter_qos>
          ...
        </datawriter_qos>
      -->
    </qos_profile>
  </qos_library>
</dds>
```

The example **Creating a Requester with optional parameters** (p. 1004) shows how to create a **FooBarRequester** (p. 2111) using this profile.

See also

Creating a Requester with optional parameters (p. 1004)

Configuring QoS Profiles with XML (p. 971)

5.103 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. 1013) 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. 973) and **Subscription Example** (p. 974) 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`.

5.104 Conventions

This section describes the conventions used in the API documentation.

5.104.1 Unsupported Features

[Not supported (optional)] This note means that the optional feature from the DDS specification is not supported in the current release.

5.104.2 API Naming Conventions

5.104.2.1 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 **DDS_Entity** (p. 1360) subtypes, the **DDS_Condition** (p. 1370) subtypes, and **DDS_WaitSet** (p. 1371). 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 type structures are made more explicit through the use of C's structure tag syntax. For example, a **DDS_↔Duration_t** (p. 1767) object must be declared as being of type "struct DDS_Duration_t," not simply "DDS_Duration_t." Interface types, by contrast, are always of typedef'd types; their underlying representations are opaque.

5.104.3 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. 1221) do not show up in the compound list or indices. This is a known limitation in the generated HTML.

5.104.4 Stereotypes

Commonly used stereotypes in the API documentation include the following.

5.104.4.1 Basic QoS

- `<<basic>>` (p. 783)
 - A QoS that Connex considers Basic.
 - See **Basic QoS** (p. 783) for a full list.

5.104.4.2 Extensions

- `<<spec>>` (p. 790)
 - Part of a DDS standard specification.
 - See **DDS Specification QoS** (p. 790) for a full list of the QoS policies and fields that are part of a DDS standard specification.
- `<<extension>>` (p. 1012)
 - An RTI Connex 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. 795) for a full list of QoS policies and fields that are extensions.

5.104.4.3 Experimental

- `<<experimental>>` (p. 1012)
 - RTI Connex 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 by RTI Connex
 - 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.

5.104.4.4 Types

- `<<interface>>` (p. 1013)
 - Pure interface type with *no state*.
 - Languages such as Java natively support the concept of an *interface* type, which is a collection of function 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. 1013)
 - 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. 2113), **FooDataType**, **FooDataWriter** (p. 2113), and **FooDataReader** (p. 2112).
 - 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. 1013)
 - Singleton class. There is a single instance of the class.
 - Generally accessed via a `get_instance()` static function.

5.104.4.5 Method Parameters

- `<<in>>` (p. 1013)
 - An *input* parameter.
- `<<out>>` (p. 1013)
 - An *output* parameter.
- `<<inout>>` (p. 1013)
 - An *input* and *output* parameter.

5.105 RTI Connex DDS API Reference

RTI Connex modules following the DDS module definitions.

Modules

- **Domain Module**

Contains the **DDS_DomainParticipant** (p. 77) class that acts as an entrypoint of RTI Connex and acts as a factory for many of the classes. The **DDS_DomainParticipant** (p. 77) also acts as a container for the other objects that make up RTI Connex.

- **Topic Module**

Contains the **DDS_Topic** (p. 175), **DDS_ContentFilteredTopic** (p. 177), and **DDS_MultiTopic** (p. 185) classes, the **DDS_TopicListener** (p. 2041) interface, and more generally, all that is needed by an application to define **DDS_Topic** (p. 175) objects and attach QoS policies to them.

- **Publication Module**

Contains the **DDS_FlowController** (p. 559), **DDS_Publisher** (p. 443), and **DDS_DataWriter** (p. 485) classes as well as the **DDS_PublisherListener** (p. 1917) and **DDS_DataWriterListener** (p. 1644) interfaces, and more generally, all that is needed on the publication side.

- **Subscription Module**

Contains the **DDS_Subscriber** (p. 572), **DDS_DataReader** (p. 615), **DDS_ReadCondition** (p. 694), **DDS_QueryCondition** (p. 699), and **DDS_TopicQuery** (p. 707) classes, as well as the **DDS_SubscriberListener** (p. 2009) and **DDS_DataReaderListener** (p. 1593) 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 Connex pluggable transports.

- **Queries and Filters Syntax**

- **Logging and Version**

APIs of troubleshooting utilities and APIs designed to configure the overall behavior of RTI Connex.

- **General Utilities and Compliance Configuration**

API of general utilities used in the RTI Connex distribution.

- **Observability Framework**

RTI Connex Observability Framework modules and APIs.

- **Durability and Persistence**

APIs related to RTI Connex Durability and Persistence.

- **System Properties**

System Properties.

- **Configuring QoS Profiles with XML**

APIs related to XML QoS Profiles.

- **Additional RTI Connex Communication Patterns**

Extensions to the RTI Connex publish-subscribe functionality.

5.105.1 Detailed Description

RTI Connex modules following the DDS module definitions.

5.105.2 Overview

Information flows with the aid of the following constructs: **DDS_Publisher** (p. 443) and **DDS_DataWriter** (p. 485) on the sending side, **DDS_Subscriber** (p. 572) and **DDS_DataReader** (p. 615) on the receiving side.

- A **DDS_Publisher** (p. 443) is an object responsible for data distribution. It may publish data of different data types. A **TDataWriter** acts as a *typed* (i.e. each **DDS_DataWriter** (p. 485) object is dedicated to one application data type) accessor to a publisher. A **DDS_DataWriter** (p. 485) 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 **DDS_Subscriber** (p. 572) 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.

DDS_Topic (p. 175) objects conceptually fit between publications and subscriptions. Publications must be known in such a way that subscriptions can refer to them unambiguously. A **DDS_Topic** (p. 175) 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 **DDS_DataWriter** (p. 485) associated with that Topic and the QoS of the **DDS_Publisher** (p. 443) associated to the **DDS_DataWriter** (p. 485) control the behavior on the publisher's side, while the corresponding **DDS_Topic** (p. 175), **DDS_DataReader** (p. 615) and **DDS_Subscriber** (p. 572) QoS control the behavior on the subscriber's side.

When an application wishes to publish data of a given type, it must create a **DDS_Publisher** (p. 443) (or reuse an already created one) and a **DDS_DataWriter** (p. 485) with all the characteristics of the desired publication. Similarly, when an application wishes to receive data, it must create a **DDS_Subscriber** (p. 572) (or reuse an already created one) and a **DDS_DataReader** (p. 615) to define the subscription.

5.105.3 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 **DDS_Entity** (p. 1360) supports its own specialized kind of QoS policies (see **All QoS Policies** (p. 1238)).
- Accepting a **DDS_Listener** (p. 1816); 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 **DDS_Entity** (p. 1360) supports its own specialized kind of listener. Listeners are related to changes in status conditions (see **Status Kinds** (p. 1222)).

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 **DDS_StatusCondition** (p. 1371) (and a set of **DDS_ReadCondition** (p. 694) objects for the **DDS_Reader** (p. 615)); conditions (in conjunction with **DDS_WaitSet** (p. 1371) 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 **DDS_DomainParticipant** (p. 77). 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.

DDS_DomainEntity (p. 1363) 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 **DDS_DataWriter** (p. 485) objects attached to the same **DDS_Publisher** (p. 443) and retrieved among **DDS_DataReader** (p. 615) objects attached to the same **DDS_Subscriber** (p. 572).

By definition, a **DDS_Topic** (p. 175) corresponds to a single data type. However, several topics may refer to the same data type. Therefore, a **DDS_Topic** (p. 175) 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. 2109).

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 **DDS_Topic** (p. 175) 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 **DDS_DomainParticipant** (p. 77).

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 **DDS_DataWriter** (p. 485) (this, in turn, will trigger its **DDS_Publisher** (p. 443)).

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 **DDS_Listener** (p. 1816) is used to provide a callback for synchronous access and a **DDS_WaitSet** (p. 1371) associated with one or several **DDS_Condition** (p. 1370) 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. 1222)). 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. 165) that the application can access as plain application data, using built-in data-readers.

5.105.4 Modules

DCPS consists of five modules:

- **Infrastructure module** (p. 716) 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. 26) contains the **DDS_DomainParticipant** (p. 77) class that acts as an entrypoint of the Service and acts as a factory for many of the classes. The **DDS_DomainParticipant** (p. 77) also acts as a container for the other objects that make up the Service.
- **Topic module** (p. 167) contains the **DDS_Topic** (p. 175) class, the **DDS_TopicListener** (p. 2041) interface, and more generally, all that is needed by the application to define **DDS_Topic** (p. 175) objects and attach QoS policies to them.
- **Publication module** (p. 438) contains the **DDS_Publisher** (p. 443) and **DDS_DataWriter** (p. 485) classes as well as the **DDS_PublisherListener** (p. 1917) and **DDS_DataWriterListener** (p. 1644) interfaces, and more generally, all that is needed on the publication side.
- **Subscription module** (p. 567) contains the **DDS_Subscriber** (p. 572), **DDS_DataReader** (p. 615), **DDS_ReadCondition** (p. 694), and **DDS_QueryCondition** (p. 699) classes, as well as the **DDS_SubscriberListener** (p. 2009) and **DDS_DataReaderListener** (p. 1593) interfaces, and more generally, all that is needed on the subscription side.

5.106 Additional RTI Connex Communication Patterns

Extensions to the RTI Connex publish-subscribe functionality.

Modules

- **Request-Reply Pattern**
Support for the request-reply communication pattern.
- **Utilities**
Utilities for the RTI Connex Messaging module.

5.106.1 Detailed Description

Extensions to the RTI Connex publish-subscribe functionality.

This section includes the **Request-Reply API** (p. 745).

5.107 Programming How-To's

These "How To"s illustrate how to apply RTI Connex API's 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 .

5.107.1 Detailed Description

These "How To"s illustrate how to apply RTI Connex API's 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.

5.108 Interface

Abstraction of a Transport Plugin network interface.

Data Structures

- 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.

5.108.1 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. 2120) **NDDS_Transport_ClassId_t** (p. 1026)

5.108.2 Enumeration Type Documentation

5.108.2.1 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.

5.109 Transport Plugins Configuration

Transport plugins configuration with RTI Connex.

Data Structures

- 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. 2164).*
- #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. 2181).*
- #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. 2181).*
- #define **NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT**
*The constant used as default value for the struct **TransportAllocationSettings_t** (p. 2181).*
- #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 Connex 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 Connex 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 Connex RTPS ports.
- **typedef RTI_INT32 NDDS_Transport_ClassId_t**
Type for storing RTI Connex Transport Plugin class IDs.

5.109.1 Detailed Description

Transport plugins configuration with RTI Connex.

Transport plugins are configured using properties. Each transport plugin must derive its property from a base configuration structure **NDDS_Transport_Property_t** (p. 2122).

To see how to configure the Built-in Transport Plugins, see **Built-in Transport Plugins** (p. 736).

See also

Built-in Transport Plugins (p. 736)

5.109.2 Macro Definition Documentation

5.109.2.1 NDDS_TRANSPORT_PORT_INVALID

```
#define NDDS_TRANSPORT_PORT_INVALID (( NDDS_Transport_Port_t) 0)
```

Port 0 is considered to be invalid.

5.109.2.2 NDDS_TRANSPORT_UUID_SIZE

```
#define NDDS_TRANSPORT_UUID_SIZE 12
```

Size of a **NDDS_Transport_UUID** (p. 2164).

5.109.2.3 NDDS_TRANSPORT_LENGTH_UNLIMITED

```
#define NDDS_TRANSPORT_LENGTH_UNLIMITED -1
```

Represent an unlimited length.

5.109.2.4 NDDS_TRANSPORT_INTERFACE_RANK_UNKNOWN

```
#define NDDS_TRANSPORT_INTERFACE_RANK_UNKNOWN 0
```

Rank interface as unknown or not yet set.

5.109.2.5 NDDS_TRANSPORT_UUID_UNKNOWN

```
#define NDDS_TRANSPORT_UUID_UNKNOWN
```

Value:

```
{ \
{ \
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \
} \
}
```

Value for UUIDs that have no known value. Used as default.

5.109.2.6 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. 2181).

5.109.2.7 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. 2181).

Automatic means the buffer size will double at every reallocation.

5.109.2.8 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. 2181).

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. 1022)
- incremental_count = **NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT_AUTOMATIC**

5.109.2.9 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.

5.109.2.10 NDDS_TRANSPORT_CLASSID_UDPv4

```
#define NDDS_TRANSPORT_CLASSID_UDPv4 (1)
```

Builtin IPv4 UDP/IP Transport Plugin class ID.

5.109.2.11 NDDS_TRANSPORT_CLASSID_SHMEM

```
#define NDDS_TRANSPORT_CLASSID_SHMEM (0x01000000)
```

Builtin Shared-Memory Transport Plugin class ID.

5.109.2.12 NDDS_TRANSPORT_CLASSID_SHMEM_510

```
#define NDDS_TRANSPORT_CLASSID_SHMEM_510 (2)
```

Builtin Shared-Memory Transport Plugin class ID for Connex 5.1.0 and earlier.

5.109.2.13 NDDS_TRANSPORT_CLASSID_UDPv6

```
#define NDDS_TRANSPORT_CLASSID_UDPv6 (2)
```

Builtin IPv6 UDP/IP Transport Plugin class ID.

5.109.2.14 NDDS_TRANSPORT_CLASSID_UDPv6_510

```
#define NDDS_TRANSPORT_CLASSID_UDPv6_510 (5)
```

Builtin IPv6 UDP/IP Transport Plugin class ID for Connex 5.1.0 and earlier.

5.109.2.15 NDDS_TRANSPORT_CLASSID_TCPV4_LAN

```
#define NDDS_TRANSPORT_CLASSID_TCPV4_LAN (8)
```

IPv4 TCP/IP Transport Plugin class ID for LAN case.

5.109.2.16 NDDS_TRANSPORT_CLASSID_TCPV4_WAN

```
#define NDDS_TRANSPORT_CLASSID_TCPV4_WAN (9)
```

IPv4 TCP/IP Transport Plugin class ID for WAN case.

5.109.2.17 NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN

```
#define NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN "tcpv4_wan"
```

IPv4 TCP/IP Transport Plugin class name for WAN case.

5.109.2.18 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.

5.109.2.19 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.

5.109.2.20 NDDS_TRANSPORT_CLASSID_UDPv4_WAN

```
#define NDDS_TRANSPORT_CLASSID_UDPv4_WAN (0x01000001)
```

Builtin IPv4 UDP/IP Asymmetric Transport Plugin class ID.

5.109.2.21 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.

5.109.2.22 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 Connexx will not need to allocate storage for a message that it retrieves with the `receive_rEA()` call.

5.109.2.23 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. 2122) structure to be valid, the value of `NDDS_Transport_Property_t::gather_send_buffer_count_max` (p. 2124) must be greater than or equal to this value.

5.109.3 Typedef Documentation

5.109.3.1 NDDS_Transport_Port_t

```
typedef RTI_UINT32 NDDS_Transport_Port_t
```

Type for storing RTI Connexx RTPS ports.

Unlike IPv4 Socket API ports, which are 2 bytes long, the RTI Connexx representation of an RTPS port is 4 bytes.

5.109.3.2 NDDS_Transport_ClassId_t

```
typedef RTI_INT32 NDDS_Transport_ClassId_t
```

Type for storing RTI Connex 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. 1025).

5.110 Transport Address

Transport-independent addressing scheme using IPv6 presentation strings and numerically stored in network-ordered format.

Data Structures

- 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. 1029).*

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.

5.110.1 Detailed Description

Transport-independent addressing scheme using IPv6 presentation strings and numerically stored in network-ordered format.

The APIs of RTI Connex 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.

5.110.2 Macro Definition Documentation

5.110.2.1 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.2120) address = NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER;

5.110.2.2 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.1029).

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,FF,FF},p=255.255.255.255:65555:65555

5.110.3 Function Documentation

5.110.3.1 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.1029) characters long.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) The address to be converted.
<i>buffer_inout</i>	<< <i>inout</i> >> (p. 1013) Storage passed in which to return the string corresponding to the address.
<i>buffer_size↔ _in</i>	<< <i>in</i> >> (p. 1013) The size of the storage buffer. Must be >= NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE (p. 1029)

Returns

1 upon success; 0 upon failure (not enough space in the provided buffer)

5.110.3.2 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. 1029) characters long.

Parameters

<i>me</i>	<< <i>in</i> >> (p. 1013) The address to be converted.
<i>buffer</i>	<< <i>inout</i> >> (p. 1013) Storage passed in which to return the string corresponding to the address.
<i>buffer_size↔ _in</i>	<< <i>in</i> >> (p. 1013) The size of the storage buffer. Must be >= NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE (p. 1029)
<i>family</i>	<< <i>in</i> >> (p. 1013) 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)

5.110.3.3 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

<i>address_out</i>	<< <i>out</i> >> (p. 1013) Numerical value of the address.
<i>address_in</i>	<< <i>in</i> >> (p. 1013) 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.

5.110.3.4 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

<i>address_in</i>	<< <i>in</i> >> (p. 1013) Address to be printed.
<i>desc_in</i>	<< <i>in</i> >> (p. 1013) A prefix to be printed before the address.
<i>indent_in</i>	<< <i>in</i> >> (p. 1013) Indentation level for the printout.

5.110.3.5 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

<i>address</i> ↔ _in	<< <i>in</i> >> (p. 1013) Address to be tested.

Note

May be implemented as a macro for efficiency.

Returns

1 if address is an IPv4 address, 0 otherwise.

5.110.3.6 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

<i>address</i> ↔ _in	<< <i>in</i> >> (p. 1013) Address to be tested.

May be implemented as a macro for efficiency.

Returns

1 if address is a multicast address, 0 otherwise.

5.110.4 Variable Documentation**5.110.4.1 NDDS_TRANSPORT_ADDRESS_INVALID**

```
const  NDDS_Transport_Address_t NDDS_TRANSPORT_ADDRESS_INVALID  [extern]
```

An invalid transport address.

5.111 UDP Transport Plugin definitions

UDP Transport Plugin definitions.

Data Structures

- 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. 2124).
- #define **NDDS_TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT** (16)
Default value of `NDDS_Transport_Property_t::gather_send_buffer_count_max` (p. 2124).
- #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.

5.111.1 Detailed Description

UDP Transport Plugin definitions.

5.111.2 Macro Definition Documentation

5.111.2.1 NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT

```
#define NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT (0)
```

Default value of **NDDS_Transport_Property_t::properties_bitmap** (p. 2124).

5.111.2.2 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. 2124).

This is also the maximum value that can be used when instantiating the udp transport.

16 is sufficient for RTI Connex, but more may improve discovery and reliable performance. Porting note: find out what the maximum gather buffer count is on your OS!

5.111.2.3 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 Connex 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. 1034) or **NDDS_Transport_UDP_Plugin::NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT** (p. 1035), depending on the property being set. See the Platform Notes to determine if this property is supported on your platform.

This value is -2.

5.111.2.4 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 Connex will automatically pick up that change.

This value is -1.

5.111.2.5 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).

5.111.2.6 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).

5.111.2.7 NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT

```
#define NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT (1)
```

Default value of multicast_ttl.

5.111.3 Typedef Documentation

5.111.3.1 NDDS_Transport_UDP_Port

```
typedef RTI_UINT16 NDDS_Transport_UDP_Port
```

UDP port.

5.112 Shared Memory Transport

Built-in transport plug-in for inter-process communications using shared memory (**NDDS_TRANSPORT_CLASSID_↔SHMEM** (p. 1024)) .

Data Structures

- struct **NDDS_Transport_Shmem_Property_t**

*Subclass of **NDDS_Transport_Property_t** (p. 2122) 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. 2123).
- `#define NDDS_TRANSPORT_SHMEM_PROPERTIES_BITMAP_DEFAULT (NDDS_TRANSPORT_↵
PROPERTY_BIT_BUFFER_ALWAYS_LOANED)`
Default value of `NDDS_Transport_Property_t::properties_bitmap` (p. 2124).
- `#define NDDS_TRANSPORT_SHMEM_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT (1024)`
Default value of `NDDS_Transport_Property_t::gather_send_buffer_count_max` (p. 2124).
- `#define NDDS_TRANSPORT_SHMEM_MESSAGE_SIZE_MAX_DEFAULT (65536)`
Default value of `NDDS_Transport_Property_t::message_size_max` (p. 2124).
- `#define NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT (64)`
Default value of `NDDS_Transport_Shmem_Property_t::received_message_count_max` (p. 2130).
- `#define NDDS_TRANSPORT_SHMEM_RECEIVE_BUFFER_SIZE_DEFAULT`
Default value of `NDDS_Transport_Shmem_Property_t::receive_buffer_size` (p. 2130).
- `#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.

5.112.1 Detailed Description

Built-in transport plug-in for inter-process communications using shared memory (`NDDS_TRANSPORT_CLASSID_↵
SHMEM` (p. 1024)) .

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. 1334).

5.112.2 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.

5.112.3 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 Connex 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](#).

5.112.4 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`

where the `port` is a function of the `domain_id` and the `participant_id`, as described in [DDS_WireProtocolQosPolicy::participant_id](#) (p. 2097)

See also

[DDS_WireProtocolQosPolicy::participant_id](#) (p. 2097)

[NDDS_Transport_Support_set_builtin_transport_property\(\)](#) (p. 735)

5.112.5 Creating and Registering Shared Memory Transport Plugin

RTI Connex can implicitly create this plugin and register with the [DDS_DomainParticipant](#) (p. 77) if this transport is specified in [DDS_TransportBuiltinQosPolicy](#) (p. 2051).

To specify the properties of the builtin shared memory transport that is implicitly registered, you can either:

- call [NDDS_Transport_Support_set_builtin_transport_property](#) (p. 735) or
- specify the pre-defined property names in [DDS_PropertyQosPolicy](#) (p. 1901) associated with the [DDS_DomainParticipant](#) (p. 77). (see [Shared Memory Transport Property Names in Property QoS Policy of Domain Participant](#) (p. 1038)).

Builtin transport plugin properties specified in [DDS_PropertyQosPolicy](#) (p. 1901) always overwrite the ones specified through [NDDS_Transport_Support_set_builtin_transport_property\(\)](#) (p. 735). 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 Connex. See [Built-in Transport Plugins](#) (p. 736) for details on when a builtin transport is registered.

To explicitly create an instance of this plugin, [NDDS_Transport_Shmem_new\(\)](#) (p. 1040) should be called. The instance should be registered with RTI Connex, see [NDDS_Transport_Support_register_transport](#) (p. 730). In some configurations, you may have to disable the builtin shared memory transport plugin instance ([DDS_TransportBuiltinQosPolicy](#) (p. 2051), [DDS_TRANSPORTBUILTIN_SHMEM](#) (p. 1334)), to avoid port conflicts with the newly created plugin instance.

5.112.6 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. 1730) to configure the builtin shared memory transport plugin.

Table 5.815 Property Strings for Shared Memory Transport

Name	Descriptions
<code>dds.transport.shmem.builtin.parent.address_bit_count</code>	See <code>NDDS_Transport_Property_t::address_bit_count</code> (p. 2123)
<code>dds.transport.shmem.builtin.parent.properties_bitmap</code>	See <code>NDDS_Transport_Property_t::properties_bitmap</code> (p. 2124)
<code>dds.transport.shmem.builtin.parent.gather_send_buffer_count_max</code>	See <code>NDDS_Transport_Property_t::gather_send_buffer_count_max</code> (p. 2124)
<code>dds.transport.shmem.builtin.parent.message_size_max</code>	See <code>NDDS_Transport_Property_t::message_size_max</code> (p. 2124)
<code>dds.transport.shmem.builtin.parent.allow_interfaces</code>	See <code>NDDS_Transport_Property_t::allow_interfaces_list</code> (p. 2125) and <code>NDDS_Transport_Property_t::allow_interfaces_list_length</code> (p. 2125). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
<code>dds.transport.shmem.builtin.parent.deny_interfaces</code>	See <code>NDDS_Transport_Property_t::deny_interfaces_list</code> (p. 2126) and <code>NDDS_Transport_Property_t::deny_interfaces_list_length</code> (p. 2126). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
<code>dds.transport.shmem.builtin.parent.allow_multicast_interfaces</code>	See <code>NDDS_Transport_Property_t::allow_multicast_interfaces_list</code> (p. 2126) and <code>NDDS_Transport_Property_t::allow_multicast_interfaces_list_length</code> (p. 2127). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
<code>dds.transport.shmem.builtin.parent.deny_multicast_interfaces</code>	See <code>NDDS_Transport_Property_t::deny_multicast_interfaces_list</code> (p. 2127) and <code>NDDS_Transport_Property_t::deny_multicast_interfaces_list_length</code> (p. 2127). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
<code>dds.transport.shmem.builtin.parent.max_interface_count</code>	See <code>NDDS_Transport_Property_t::max_interface_count</code> (p. 2128)
<code>dds.transport.shmem.builtin.parent.thread_name_prefix</code>	See <code>NDDS_Transport_Property_t::thread_name_prefix</code> (p. 2128)
<code>dds.transport.shmem.builtin.received_message_count_max</code>	See <code>NDDS_Transport_Shmem_Property_t::received_message_count_max</code> (p. 2130)

Name	Descriptions
dds.transport.shmem.builtin.receive_buffer_size	See <code>NDDS_Transport_Shmem_Property_t::receive_buffer_size</code> (p. 2130)
dds.transport.shmem.builtin.enable_udp_debugging	See <code>NDDS_Transport_Shmem_Property_t::enable_udp_debugging</code> (p. 2131)
dds.transport.shmem.builtin.udp_debugging_address	See <code>NDDS_Transport_Shmem_Property_t::udp_debugging_address</code> (p. 2131)
dds.transport.shmem.builtin.udp_debugging_port	See <code>NDDS_Transport_Shmem_Property_t::udp_debugging_port</code> (p. 2132)

5.112.7 Macro Definition Documentation

5.112.7.1 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. 2123).

5.112.7.2 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. 2124).

5.112.7.3 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. 2124).

5.112.7.4 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. 2124).

5.112.7.5 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. 2130).

5.112.7.6 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. 2130).

5.112.7.7 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)

5.112.7.8 NDDS_TRANSPORT_SHMEM_PROPERTY_DEFAULT

```
#define NDDS_TRANSPORT_SHMEM_PROPERTY_DEFAULT
```

Use this to initialize stack variable.

5.112.8 Function Documentation

5.112.8.1 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

<i>property_in</i>	<< <i>in</i> >> (p. 1013) 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.

5.112.8.2 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

<i>default_network_address_out</i>	<< <i>out</i> >> (p. 1013) Network address to be used when registering the transport.
<i>property_in</i>	<< <i>in</i> >> (p. 1013) 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.
<i>minimum_compatibility_version</i>	<< <i>in</i> >> (p. 1013) 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 <i>minimum_compatibility_version</i> property for more information.

Returns

handle to a Shmem inter-process Transport Plugin on success
 NULL on failure.

5.113 UDPv4 Transport

Transport plug-in using UDP/IPv4 (**NDDS_TRANSPORT_CLASSID_UDPv4** (p. 1023)) .

Data Structures

- 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. 2123).
- #define **NDDS_TRANSPORT_UDPv4_WAN_ADDRESS_BIT_COUNT** (128)
Default value of `NDDS_Transport_Property_t::address_bit_count` (p. 2123) 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. 2124).
- #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. 2124).
- #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. 2135) and `NDDS_Transport_↵`
`UDPv4_WAN_Property_t::send_socket_buffer_size` (p. 2145).*
- #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. 2136) and `NDDS_Transport_↵`
`UDPv4_WAN_Property_t::recv_socket_buffer_size` (p. 2146).*
- #define **NDDS_TRANSPORT_UDPv4_PAYLOAD_SIZE_MAX** (65507)
Maximum value of `NDDS_Transport_Property_t::message_size_max` (p. 2124).
- #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. 2124).
- #define **NDDS_TRANSPORT_UDPv4_MULTICAST_TTL_DEFAULT** **NDDS_TRANSPORT_UDP_↵**
MULTICAST_TTL_DEFAULT
Default value of `NDDS_Transport_UDPv4_Property_t::multicast_ttl` (p. 2137).
- #define **NDDS_TRANSPORT_UDPv4_BLOCKING_NEVER**
Value for `NDDS_Transport_UDPv4_Property_t::send_blocking` (p. 2139) to specify non-blocking sockets.
- #define **NDDS_TRANSPORT_UDPv4_BLOCKING_ALWAYS**
[default] Value for `NDDS_Transport_UDPv4_Property_t::send_blocking` (p. 2139) 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. 2139) to specify blocking sockets.
- #define **NDDS_TRANSPORT_UDPv4_PROPERTY_DEFAULT**
Use this to initialize a `NDDS_Transport_UDPv4_Property_t` (p. 2133) 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.

5.113.1 Detailed Description

Transport plug-in using UDP/IPv4 (**DDS_TRANSPORT_CLASSID_UDPv4** (p. 1023)) .

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. 1334).

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. 2136) and **NDDS_Transport_UDPv4_Property_t::multicast_enabled** (p. 2137).

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. 2128) and the "white" and "black" lists in the base property's fields (**NDDS_Transport_Property_t::allow_interfaces_list** (p. 2125), **NDDS_Transport_Property_t::deny_interfaces_list** (p. 2126), **NDDS_Transport_Property_t::allow_multicast_interfaces_list** (p. 2126), **NDDS_Transport_Property_t::deny_multicast_interfaces_list** (p. 2127)).

RTI Connexant can implicitly create this plugin and register with the **DDS_DomainParticipant** (p. 77) if this transport is specified in **DDS_TransportBuiltinQosPolicy** (p. 2051).

To specify the properties of the builtin UDPv4 transport that is implicitly registered, you can either:

- call **NDDS_Transport_Support_set_builtin_transport_property** (p. 735) or
- specify the predefined property names in **DDS_PropertyQosPolicy** (p. 1901) associated with the **DDS_DomainParticipant** (p. 77). (see **UDPv4 Transport Property Names in Property QoS Policy of Domain Participant** (p. 1044)). Builtin transport plugin properties specified in **DDS_PropertyQosPolicy** (p. 1901) always overwrite the ones specified through **NDDS_Transport_Support_set_builtin_transport_property()** (p. 735). 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 Connexant. Any properties set after the builtin transport is registered will be ignored. See **Built-in Transport Plugins** (p. 736) for details on when a builtin transport is registered. To explicitly create an instance of this plugin, **NDDS_Transport_UDPv4_new()** (p. 1049) should be called. The instance should be registered with RTI Connexant, see **NDDS_Transport_Support_register_transport** (p. 730). In some configurations one may have to disable the builtin UDPv4 transport plugin instance (**DDS_TransportBuiltinQosPolicy** (p. 2051), **DDS_TRANSPORTBUILTIN_UDPv4** (p. 1334)), to avoid port conflicts with the newly created plugin instance.

5.113.2 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. 1730) to configure the builtin UDPv4 transport plugin.

Table 5.818 Property Names for UDPv4 Transport Plugin

Property Name	Description
<code>dds.transport.UDPv4.builtin.parent.classid</code>	See <code>NDDS_Transport_Property_t::classid</code> (p. 2123) Should be set to " <code>NDDS_TRANSPORT_CLASSID_↵ UDPv6</code> (p. 1023)"
<code>dds.transport.UDPv4.builtin.parent.address_bit_count</code>	See <code>NDDS_Transport_Property_t::address_bit_↵ count</code> (p. 2123)
<code>dds.transport.UDPv4.builtin.parent.properties_bitmap</code>	See <code>NDDS_Transport_Property_t::properties_↵ bitmap</code> (p. 2124)
<code>dds.transport.UDPv4.builtin.parent.gather_send_↵ buffer_count_max</code>	See <code>NDDS_Transport_Property_t::gather_send_↵ buffer_count_max</code> (p. 2124)
<code>dds.transport.UDPv4.builtin.parent.message_size_max</code>	See <code>NDDS_Transport_Property_t::message_size_↵ max</code> (p. 2124)
<code>dds.transport.UDPv4.builtin.parent.allow_interfaces</code>	See <code>NDDS_Transport_Property_t::allow_↵ interfaces_list</code> (p. 2125) and <code>NDDS_Transport_↵ Property_t::allow_interfaces_list_length</code> (p. 2125). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example, 127.0.0.1,eth0
<code>dds.transport.UDPv4.builtin.parent.deny_interfaces</code>	See <code>NDDS_Transport_Property_t::deny_interfaces_↵ _list</code> (p. 2126) and <code>NDDS_Transport_Property_t_↵ ::deny_interfaces_list_length</code> (p. 2126). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
<code>dds.transport.UDPv4.builtin.parent.allow_multicast_↵ interfaces</code>	See <code>NDDS_Transport_Property_t::allow_multicast_↵ _interfaces_list</code> (p. 2126) and <code>NDDS_Transport_↵ Property_t::allow_multicast_interfaces_list_length</code> (p. 2127). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
<code>dds.transport.UDPv4.builtin.parent.deny_multicast_↵ interfaces</code>	See <code>NDDS_Transport_Property_t::deny_multicast_↵ _interfaces_list</code> (p. 2127) and <code>NDDS_Transport_↵ Property_t::deny_multicast_interfaces_list_length</code> (p. 2127). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
<code>dds.transport.UDPv4.builtin.parent.max_interface_count</code>	See <code>NDDS_Transport_Property_t::max_interface_↵ count</code> (p. 2128)
<code>dds.transport.UDPv4.builtin.parent.thread_name_prefix</code>	See <code>NDDS_Transport_Property_t::thread_name_↵ prefix</code> (p. 2128)
<code>dds.transport.UDPv4.builtin.send_socket_buffer_size</code>	See <code>NDDS_Transport_UDPv4_Property_t::send_↵ socket_buffer_size</code> (p. 2135)
<code>dds.transport.UDPv4.builtin.recv_socket_buffer_size</code>	See <code>NDDS_Transport_UDPv4_Property_t::recv_↵ socket_buffer_size</code> (p. 2136)

Property Name	Description
dds.transport.UDPv4.builtin.unicast_enabled	See <code>NDDS_Transport_UDPv4_Property_t::unicast_↵_enabled</code> (p. 2136)
dds.transport.UDPv4.builtin.multicast_enabled	See <code>NDDS_Transport_UDPv4_Property_t::↵multicast_enabled</code> (p. 2137)
dds.transport.UDPv4.builtin.multicast_ttl	See <code>NDDS_Transport_UDPv4_Property_t::↵multicast_ttl</code> (p. 2137)
dds.transport.UDPv4.builtin.multicast_loopback_↵disabled	See <code>NDDS_Transport_UDPv4_Property_t::↵multicast_loopback_disabled</code> (p. 2137)
dds.transport.UDPv4.builtin.ignore_loopback_interface	See <code>NDDS_Transport_UDPv4_Property_t::↵ignore_loopback_interface</code> (p. 2137)
dds.transport.UDPv4.builtin.ignore_nonrunning_↵interfaces	See <code>NDDS_Transport_UDPv4_Property_t::↵ignore_nonrunning_interfaces</code> (p. 2138)
dds.transport.UDPv4.builtin.ignore_nonup_interfaces	[DEPRECATED] See <code>NDDS_Transport_UDPv4_Property_t::↵ignore_nonup_interfaces</code> (p. 2138)
dds.transport.UDPv4.builtin.no_zero_copy	[DEPRECATED] See <code>NDDS_Transport_UDPv4_Property_t::↵no_zero_copy</code> (p. 2139)
dds.transport.UDPv4.builtin.send_blocking	See <code>NDDS_Transport_UDPv4_Property_t::↵send_blocking</code> (p. 2139)
dds.transport.UDPv4.builtin.transport_priority_mask	See <code>NDDS_Transport_UDPv4_Property_t::↵transport_priority_mask</code> (p. 2140)
dds.transport.UDPv4.builtin.transport_priority_↵mapping_low	See <code>NDDS_Transport_UDPv4_Property_t::↵transport_priority_mapping_low</code> (p. 2140)
dds.transport.UDPv4.builtin.transport_priority_↵mapping_high	See <code>NDDS_Transport_UDPv4_Property_t::↵transport_priority_mapping_high</code> (p. 2140)
dds.transport.UDPv4.builtin.send_ping	See <code>NDDS_Transport_UDPv4_Property_t::↵send_ping</code> (p. 2141)
dds.transport.UDPv4.builtin.force_interface_poll_↵detection	See <code>NDDS_Transport_UDPv4_Property_t::↵force_interface_poll_detection</code> (p. 2141)
dds.transport.UDPv4.builtin.interface_poll_period	See <code>NDDS_Transport_UDPv4_Property_t::↵interface_poll_period</code> (p. 2141)
dds.transport.UDPv4.builtin.reuse_multicast_receive_↵resource	See <code>NDDS_Transport_UDPv4_Property_t::↵reuse_multicast_receive_resource</code> (p. 2141)
dds.transport.UDPv4.builtin.protocol_overhead_max	See <code>NDDS_Transport_UDPv4_Property_t::↵protocol_overhead_max</code> (p. 2142)
dds.transport.UDPv4.builtin.disable_interface_tracking	See <code>NDDS_Transport_UDPv4_Property_t::↵disable_interface_tracking</code> (p. 2142)
dds.transport.UDPv4.builtin.public_address	See <code>NDDS_Transport_UDPv4_Property_t::↵public_address</code> (p. 2143)
dds.transport.UDPv4.builtin.join_multicast_group_↵timeout	See <code>NDDS_Transport_UDPv4_Property_t::↵join_multicast_group_timeout</code> (p. 2142)
dds.transport.UDPv4.builtin.port_offset	See <code>NDDS_Transport_UDPv4_Property_t::↵port_offset</code> (p. 2143)

See also

NDDS_Transport_Support_set_builtin_transport_property() (p. 735)

5.113.3 Macro Definition Documentation

5.113.3.1 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. 2123).

5.113.3.2 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. 2123) for UDPv4 asymmetric transport.

5.113.3.3 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. 2124).

5.113.3.4 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. 2124).

This is also the maximum value that can be used when instantiating the UDP transport.

16 is sufficient for RTI Connex, but more may improve discovery and reliable performance. Porting note: find out what the maximum gather buffer count is on your OS!

5.113.3.5 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 Connexnt 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.

5.113.3.6 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.

5.113.3.7 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. 2135) and **NDDS_Transport_↔_UDPv4_WAN_Property_t::send_socket_buffer_size** (p. 2145).

The default value is 2097152 bytes (1048576 bytes on AIX platforms).

5.113.3.8 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. 2136) and **NDDS_Transport_↔_UDPv4_WAN_Property_t::recv_socket_buffer_size** (p. 2146).

The default value is 2097152 bytes (1048576 bytes on AIX platforms).

5.113.3.9 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. 2124).

5.113.3.10 **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. 2124).

5.113.3.11 **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. 2137).

5.113.3.12 **NDDS_TRANSPORT_UDPv4_BLOCKING_NEVER**

```
#define NDDS_TRANSPORT_UDPv4_BLOCKING_NEVER
```

Value for **NDDS_Transport_UDPv4_Property_t::send_blocking** (p. 2139) to specify non-blocking sockets.

5.113.3.13 **NDDS_TRANSPORT_UDPv4_BLOCKING_ALWAYS**

```
#define NDDS_TRANSPORT_UDPv4_BLOCKING_ALWAYS
```

[default] Value for **NDDS_Transport_UDPv4_Property_t::send_blocking** (p. 2139) to specify blocking sockets.

5.113.3.14 **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. 2139) to specify blocking sockets.

5.113.3.15 **NDDS_TRANSPORT_UDPv4_PROPERTY_DEFAULT**

```
#define NDDS_TRANSPORT_UDPv4_PROPERTY_DEFAULT
```

Use this to initialize a **NDDS_Transport_UDPv4_Property_t** (p. 2133) structure.

5.113.3.16 **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

<i>self</i>	NOT USED. May be NULL.
<i>address_out</i>	<< <i>out</i> >> (p. 1013) The corresponding numerical value in IPv4 format.
<i>address_in</i>	<< <i>in</i> >> (p. 1013) 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.

5.113.4 Function Documentation

5.113.4.1 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 Connex. Multiple instances of the plugin may be useful for partitioning the network interfaces across multiple RTI Connex 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. 2125),
- **NDDS_Transport_Property_t::deny_interfaces_list** (p. 2126),
- **NDDS_Transport_Property_t::allow_multicast_interfaces_list** (p. 2126),
- **NDDS_Transport_Property_t::deny_multicast_interfaces_list** (p. 2127)

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

<i>property_in</i>	<< <i>in</i> >> (p. 1013) Desired behavior of this transport. May be NULL for default property.

Returns

A UDPv4 Transport Plugin instance on success; or NULL on failure.

5.113.4.2 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 Connex. Multiple instances of the plugin may be useful for partitioning the network interfaces across multiple RTI Connex 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. 2135):

- **NDDS_Transport_Property_t::allow_interfaces_list** (p. 2125),
- **NDDS_Transport_Property_t::deny_interfaces_list** (p. 2126),
- **NDDS_Transport_Property_t::allow_multicast_interfaces_list** (p. 2126),
- **NDDS_Transport_Property_t::deny_multicast_interfaces_list** (p. 2127)

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

<i>default_network_address_out</i>	<< <i>out</i> >> (p. 1013) Network address to be used when registering the transport.
<i>property_in</i>	<< <i>in</i> >> (p. 1013) Desired behavior of this transport as defined in the DDS_DomainParticipantQos::property (p. 1730).

Returns

A UDPv4 Transport Plugin instance on success; or NULL on failure.

5.113.4.3 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

<i>default_network_address_out</i>	<< <i>out</i> >> (p. 1013) Network address to be used when registering the transport.
<i>property_in</i>	<< <i>in</i> >> (p. 1013) Desired behavior of this transport as defined in the DDS_DomainParticipantQos::property (p. 1730).
<i>propertyPrefix</i>	a prefix for the properties. Expected UDP properties have the form prefix.property_name

5.114 Real-Time WAN Transport

Transport plug-in using UDP/IPv4 for WAN communications. (**NDDS_TRANSPORT_CLASSID_UDPv4_WAN** (p. 1025))

Data Structures

- 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. 2124).*
- #define **NDDS_TRANSPORT_UDPv4_WAN_PROPERTY_DEFAULT**
*Use this to initialize a **NDDS_Transport_UDPv4_WAN_Property_t** (p. 2144) 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.

5.114.1 Detailed Description

Transport plug-in using UDP/IPv4 for WAN communications. (**NDDS_TRANSPORT_CLASSID_UDPv4_WAN** (p. 1025))

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 Connex 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 Connex 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 Connex 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. 1334).

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. 2128) and the "white" and "black" lists in the base property's fields (**NDDS_Transport_Property_t::allow_interfaces_list** (p. 2125), **NDDS_Transport_Property_t::deny_↵interfaces_list** (p. 2126)).

RTI Connext can implicitly create this plugin and register with the **DDS_DomainParticipant** (p. 77) if this transport is specified in **DDS_TransportBuiltinQosPolicy** (p. 2051).

To specify the properties of the Real-Time WAN Transport that is implicitly registered, you can either:

- call **NDDS_Transport_Support_set_builtin_transport_property** (p. 735) or
- specify the predefined property names in **DDS_PropertyQosPolicy** (p. 1901) associated with the **DDS_↵DomainParticipant** (p. 77). (see **Real-Time WAN Transport Property** (p. 1053)). Builtin transport plugin properties specified in **DDS_PropertyQosPolicy** (p. 1901) always overwrite the ones specified through **NDDS_↵Transport_Support_set_builtin_transport_property()** (p. 735). 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. 736) for details on when a builtin transport is registered. To explicitly create an instance of this plugin, **NDDS_Transport_UDPv4_↵WAN_new()** (p. 1056) should be called. The instance should be registered with RTI Connext, see **NDDS_Transport_↵Support_register_transport** (p. 730). In some configurations one may have to disable the builtin UDPv4 transport plugin instance (**DDS_TransportBuiltinQosPolicy** (p. 2051), **DDS_TRANSPORTBUILTIN_UDPv4_WAN** (p. 1334)), 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.

5.114.2 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. 1730) to configure the Real-Time WAN Transport plugin.

Table 5.823 *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. 2123) Should be set to "NDDS_TRANSPORT_CLASSID_↵ UDPv4_WAN (p. 1025)"
dds.transport.UDPv4_WAN.builtin.parent.address_bit_↵ _count	See NDDS_Transport_Property_t::address_bit_↵ count (p. 2123)
dds.transport.UDPv4_WAN.builtin.parent.properties_↵ bitmap	See NDDS_Transport_Property_t::properties_↵ bitmap (p. 2124)
dds.transport.UDPv4_WAN.builtin.parent.gather_↵ send_buffer_count_max	See NDDS_Transport_Property_t::gather_send_↵ buffer_count_max (p. 2124)
dds.transport.UDPv4_WAN.builtin.parent.message_↵ size_max	See NDDS_Transport_Property_t::message_size_↵ max (p. 2124)
dds.transport.UDPv4_WAN.builtin.parent.allow_↵ interfaces	See NDDS_Transport_Property_t::allow_↵ interfaces_list (p. 2125) and NDDS_Transport_↵ Property_t::allow_interfaces_list_length (p. 2125). 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. 2126) and NDDS_Transport_Property_t_↵ ::deny_interfaces_list_length (p. 2126). 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. 2128)
dds.transport.UDPv4_WAN.builtin.parent.thread_↵ name_prefix	See NDDS_Transport_Property_t::thread_name_↵ prefix (p. 2128)
dds.transport.UDPv4_WAN.builtin.send_socket_↵ buffer_size	See NDDS_Transport_UDPv4_WAN_Property_t_↵ ::send_socket_buffer_size (p. 2145)
dds.transport.UDPv4_WAN.builtin.recv_socket_buffer_↵ _size	See NDDS_Transport_UDPv4_WAN_Property_t_↵ ::recv_socket_buffer_size (p. 2146)
dds.transport.UDPv4_WAN.builtin.ignore_loopback_↵ interface	See NDDS_Transport_UDPv4_WAN_Property_t_↵ ::ignore_loopback_interface (p. 2146)
dds.transport.UDPv4_WAN.builtin.ignore_nonrunning_↵ _interfaces	See NDDS_Transport_UDPv4_WAN_Property_t_↵ ::ignore_nonrunning_interfaces (p. 2147)
dds.transport.UDPv4_WAN.builtin.ignore_nonup_↵ interfaces	[DEPRECATED] See NDDS_Transport_UDPv4_↵ WAN_Property_t::ignore_nonup_interfaces (p. 2146)
dds.transport.UDPv4_WAN.builtin.no_zero_copy	[DEPRECATED] See NDDS_Transport_UDPv4_↵ WAN_Property_t::no_zero_copy (p. 2147)
dds.transport.UDPv4_WAN.builtin.send_blocking	See NDDS_Transport_UDPv4_WAN_Property_t_↵ ::send_blocking (p. 2148)
dds.transport.UDPv4_WAN.builtin.transport_priority_↵ mask	See NDDS_Transport_UDPv4_WAN_Property_t_↵ ::transport_priority_mask (p. 2148)
dds.transport.UDPv4_WAN.builtin.transport_priority_↵ mapping_low	See NDDS_Transport_UDPv4_WAN_Property_t_↵ ::transport_priority_mapping_low (p. 2149)
dds.transport.UDPv4_WAN.builtin.transport_priority_↵ mapping_high	See NDDS_Transport_UDPv4_WAN_Property_t_↵ ::transport_priority_mapping_high (p. 2149)

Property Name	Description
dds.transport.UDPv4_WAN.builtin.send_ping	See <code>NDDS_Transport_UDPv4_WAN_Property_t::send_ping</code> (p. 2149)
dds.transport.UDPv4_WAN.builtin.force_interface_poll_detection	See <code>NDDS_Transport_UDPv4_WAN_Property_t::force_interface_poll_detection</code> (p. 2150)
dds.transport.UDPv4_WAN.builtin.interface_poll_period	See <code>NDDS_Transport_UDPv4_WAN_Property_t::interface_poll_period</code> (p. 2150)
dds.transport.UDPv4_WAN.builtin.protocol_overhead_max	See <code>NDDS_Transport_UDPv4_WAN_Property_t::protocol_overhead_max</code> (p. 2150)
dds.transport.UDPv4_WAN.builtin.disable_interface_tracking	See <code>NDDS_Transport_UDPv4_WAN_Property_t::disable_interface_tracking</code> (p. 2150)
dds.transport.UDPv4_WAN.builtin.public_address	See <code>NDDS_Transport_UDPv4_WAN_Property_t::public_address</code> (p. 2151)
dds.transport.UDPv4_WAN.builtin.comm_ports	See <code>NDDS_Transport_UDPv4_WAN_Property_t::comm_ports_list</code> (p. 2151)
dds.transport.UDPv4_WAN.builtin.port_offset	See <code>NDDS_Transport_UDPv4_WAN_Property_t::port_offset</code> (p. 2153)
dds.transport.UDPv4_WAN.builtin.binding_ping_period	See <code>NDDS_Transport_UDPv4_WAN_Property_t::binding_ping_period</code> (p. 2153)

See also

`NDDS_Transport_Support_set_builtin_transport_property()` (p. 735)

5.114.3 Macro Definition Documentation

5.114.3.1 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. 2124).

5.114.3.2 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. 2144) structure.

5.114.4 Function Documentation

5.114.4.1 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 Connex. Multiple instances of the plugin may be useful for partitioning the network interfaces across multiple RTI Connex 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. 2125),
- `NDDS_Transport_Property_t::deny_interfaces_list` (p. 2126),

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

<code>property</code> ↵ _in	<< <i>in</i> >> (p. 1013) Desired behavior of this transport. May be NULL for default property.

Returns

A UDPv4_WAN Transport Plugin instance on success; or NULL on failure.

5.114.4.2 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. 2145):

- **NDDS_Transport_Property_t::allow_interfaces_list** (p. 2125),
- **NDDS_Transport_Property_t::deny_interfaces_list** (p. 2126),

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

<i>default_network_address_out</i>	<< out >> (p. 1013) Network address to be used when registering the transport.
<i>property_in</i>	<< in >> (p. 1013) Desired behavior of this transport as defined in the DDS_DomainParticipantQos::property (p. 1730).

Returns

A UDPv4_WAN Transport Plugin instance on success; or NULL on failure.

5.114.4.3 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

<i>default_network_address_out</i>	<< out >> (p. 1013) Network address to be used when registering the transport.
<i>property_in</i>	<< in >> (p. 1013) Desired behavior of this transport as defined in the DDS_DomainParticipantQos::property (p. 1730).
<i>propertyPrefix</i>	Prefix for the properties. Expected UDP properties have the form prefix.property_name

5.115 UDPv6 Transport

Transport plug-in using UDP/IPv6 (**NDDS_TRANSPORT_CLASSID_UDPv6** (p. 1024)) .

Data Structures

- 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. 2123).*
- #define **NDDS_TRANSPORT_UDPv6_PROPERTIES_BITMAP_DEFAULT** **NDDS_TRANSPORT_UDP_↔
PROPERTIES_BITMAP_DEFAULT**
*Default value of **NDDS_Transport_Property_t::properties_bitmap** (p. 2124).*
- #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. 2124).*
- #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. 2155).*
- #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. 2156).*
- #define **NDDS_TRANSPORT_UDPv6_PAYLOAD_SIZE_MAX** (65487)
*Maximum value of **NDDS_Transport_Property_t::message_size_max** (p. 2124).*
- #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. 2124).*

- `#define NDDS_TRANSPORT_UDPv6_MULTICAST_TTL_DEFAULT NDDS_TRANSPORT_UDPv6_MULTICAST_TTL_DEFAULT`
Default value of `NDDS_Transport_UDPv6_Property_t::multicast_ttl` (p. 2157).
- `#define NDDS_TRANSPORT_UDPv6_BLOCKING_NEVER`
Value for `NDDS_Transport_UDPv6_Property_t::send_blocking` (p. 2159) to specify non-blocking sockets.
- `#define NDDS_TRANSPORT_UDPv6_BLOCKING_ALWAYS`
[default] Value for `NDDS_Transport_UDPv6_Property_t::send_blocking` (p. 2159) 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.

5.115.1 Detailed Description

Transport plug-in using UDP/IPv6 (`NDDS_TRANSPORT_CLASSID_UDPv6` (p. 1024)) .

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. 1334).

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. 2156) and `NDDS_Transport_UDPv6_Property_t::multicast_enabled` (p. 2157).

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. 2128) and the "white" and "black" lists in the base property's fields (`NDDS_Transport_Property_t::allow_interfaces_list` (p. 2125), `NDDS_Transport_Property_t::deny_interfaces_list` (p. 2126), `NDDS_Transport_Property_t::allow_multicast_interfaces_list` (p. 2126), `NDDS_Transport_Property_t::deny_multicast_interfaces_list` (p. 2127)).

RTI Connext can implicitly create this plugin and register it with the `DDS_DomainParticipant` (p. 77) if this transport is specified in the `DDS_TransportBuiltinQosPolicy` (p. 2051).

To specify the properties of the builtin UDPv6 transport that is implicitly registered, you can either:

- call **NDDS_Transport_Support_set_builtin_transport_property** (p. 735) or
- specify the predefined property names in **DDS_PropertyQoSPolicy** (p. 1901) associated with the **DDS_DomainParticipant** (p. 77). (see **UDPv6 Transport Property Names in Property QoS Policy of Domain Participant** (p. 1060)). Builtin transport plugin properties specified in **DDS_PropertyQoSPolicy** (p. 1901) always overwrite the ones specified through **NDDS_Transport_Support_set_builtin_transport_property()** (p. 735). 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. 736) for details on when a builtin transport is registered.

To explicitly create an instance of this plugin, **NDDS_Transport_UDPv6_new()** (p. 1065) should be called. The instance should be registered with RTI Connext, see **NDDS_Transport_Support_register_transport** (p. 730). In some configurations, you may have to disable the builtin UDPv6 transport plugin instance (**DDS_TransportBuiltinQoSPolicy** (p. 2051), **DDS_TRANSPORTBUILTIN_UDPv6** (p. 1334)), to avoid port conflicts with the newly created plugin instance.

5.115.2 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. 1901) of a **DDS_DomainParticipant** (p. 77) to configure the builtin UDPv6 transport plugin.

Table 5.827 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. 2123)
dds.transport.UDPv6.builtin.parent.properties_bitmap	See NDDS_Transport_Property_t::properties_bitmap (p. 2124)
dds.transport.UDPv6.builtin.parent.gather_send_buffer_count_max	See NDDS_Transport_Property_t::gather_send_buffer_count_max (p. 2124)
dds.transport.UDPv6.builtin.parent.message_size_max	See NDDS_Transport_Property_t::message_size_max (p. 2124)
dds.transport.UDPv6.builtin.parent.allow_interfaces	See NDDS_Transport_Property_t::allow_interfaces_list (p. 2125) and NDDS_Transport_Property_t::allow_interfaces_list_length (p. 2125). 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. 2126) and NDDS_Transport_Property_t::deny_interfaces_list_length (p. 2126). 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. 2126) and NDDS_Transport_Property_t::allow_multicast_interfaces_list_length (p. 2127). Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
Generated by Doxygen	

Property Name	Description
dds.transport.UDPv6.builtin.parent.deny_multicast_↵ interfaces	See NDDS_Transport_Property_t::deny_multicast_↵ _interfaces_list (p.2127) and NDDS_Transport_↵ Property_t::deny_multicast_interfaces_list_length (p.2127). 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. 2128)
dds.transport.UDPv6.builtin.parent.thread_name_prefix	See NDDS_Transport_Property_t::thread_name_↵ prefix (p. 2128)
dds.transport.UDPv6.builtin.send_socket_buffer_size	See NDDS_Transport_UDPv6_Property_t::send_↵ socket_buffer_size (p. 2155)
dds.transport.UDPv6.builtin.recv_socket_buffer_size	See NDDS_Transport_UDPv6_Property_t::recv_↵ socket_buffer_size (p. 2156)
dds.transport.UDPv6.builtin.unicast_enabled	See NDDS_Transport_UDPv6_Property_t::unicast_↵ _enabled (p. 2156)
dds.transport.UDPv6.builtin.multicast_enabled	See NDDS_Transport_UDPv6_Property_t_↵ ::multicast_enabled (p. 2157)
dds.transport.UDPv6.builtin.multicast_ttl	See NDDS_Transport_UDPv6_Property_t_↵ ::multicast_ttl (p. 2157)
dds.transport.UDPv6.builtin.multicast_loopback_↵ disabled	See NDDS_Transport_UDPv6_Property_t_↵ ::multicast_loopback_disabled (p. 2157)
dds.transport.UDPv6.builtin.ignore_loopback_interface	See NDDS_Transport_UDPv6_Property_t::ignore_↵ loopback_interface (p. 2157)
dds.transport.UDPv6.builtin.ignore_nonrunning_↵ interfaces	See NDDS_Transport_UDPv6_Property_t::ignore_↵ nonrunning_interfaces (p. 2158)
dds.transport.UDPv6.builtin.no_zero_copy	[DEPRECATED] See NDDS_Transport_UDPv6_↵ Property_t::no_zero_copy (p. 2158)
dds.transport.UDPv6.builtin.send_blocking	See NDDS_Transport_UDPv6_Property_t::send_↵ blocking (p. 2159)
dds.transport.UDPv6.builtin.enable_v4mapped	See NDDS_Transport_UDPv6_Property_t::enable_↵ v4mapped (p. 2159)
dds.transport.UDPv6.builtin.transport_priority_mask	See NDDS_Transport_UDPv6_Property_t_↵ ::transport_priority_mask (p. 2159)
dds.transport.UDPv6.builtin.transport_priority_↵ mapping_low	See NDDS_Transport_UDPv6_Property_t_↵ ::transport_priority_mapping_low (p. 2160)
dds.transport.UDPv6.builtin.transport_priority_↵ mapping_high	See NDDS_Transport_UDPv6_Property_t_↵ ::transport_priority_mapping_high (p. 2160)
dds.transport.UDPv6.builtin.send_ping	See NDDS_Transport_UDPv6_Property_t::send_↵ ping (p.2160) interface_poll_detection_kind
dds.transport.UDPv6.builtin.force_interface_poll_↵ detection	See NDDS_Transport_UDPv6_Property_t::force_↵ interface_poll_detection (p. 2161)
dds.transport.UDPv6.builtin.interface_poll_period	See NDDS_Transport_UDPv6_Property_t_↵ ::interface_poll_period (p. 2161)
dds.transport.UDPv6.builtin.reuse_multicast_receive_↵ resource	See NDDS_Transport_UDPv6_Property_t::reuse_↵ multicast_receive_resource (p. 2161)
dds.transport.UDPv6.builtin.protocol_overhead_max	See NDDS_Transport_UDPv6_Property_t_↵ ::protocol_overhead_max (p. 2161)

Property Name	Description
dds.transport.UDPv6.builtin.disable_interface_tracking	See NDDS_Transport_UDPv6_Property_t::disable_interface_tracking (p. 2162)
dds.transport.UDPv6.builtin.public_address	See NDDS_Transport_UDPv6_Property_t::public_address (p. 2162)
dds.transport.UDPv6.builtin.join_multicast_group_timeout	See NDDS_Transport_UDPv6_Property_t::join_multicast_group_timeout (p. 2162)
dds.transport.UDPv6.builtin.port_offset	See NDDS_Transport_UDPv6_Property_t::port_offset (p. 2163)

See also

[NDDS_Transport_Support_set_builtin_transport_property\(\)](#) (p. 735)

5.115.3 Macro Definition Documentation

5.115.3.1 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. 2123).

5.115.3.2 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. 2124).

5.115.3.3 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. 2124).

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!

5.115.3.4 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 Connex 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.

5.115.3.5 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.

5.115.3.6 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. 2155).

The default value is 2097152 bytes (1048576 bytes on AIX platforms).

5.115.3.7 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. 2156).

The default value is 2097152 bytes (1048576 bytes on AIX platforms).

5.115.3.8 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. 2124).

5.115.3.9 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. 2124).

5.115.3.10 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. 2157).

5.115.3.11 NDDS_TRANSPORT_UDPv6_BLOCKING_NEVER

```
#define NDDS_TRANSPORT_UDPv6_BLOCKING_NEVER
```

Value for **NDDS_Transport_UDPv6_Property_t::send_blocking** (p. 2159) to specify non-blocking sockets.

5.115.3.12 NDDS_TRANSPORT_UDPv6_BLOCKING_ALWAYS

```
#define NDDS_TRANSPORT_UDPv6_BLOCKING_ALWAYS
```

[default] Value for **NDDS_Transport_UDPv6_Property_t::send_blocking** (p. 2159) to specify blocking sockets.

5.115.3.13 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

<i>self</i>	NOT USED. May be NULL.
<i>address_out</i>	<< out >> (p. 1013) The corresponding numerical value in IPv6 format.
<i>address_in</i>	<< in >> (p. 1013) 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.

5.115.4 Function Documentation

5.115.4.1 `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 Connex. Multiple instances of the plugin may be useful for partitioning the network interfaces across multiple RTI Connex 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. 2155):

- **NDDS_Transport_Property_t::allow_interfaces_list** (p. 2125),
- **NDDS_Transport_Property_t::deny_interfaces_list** (p. 2126),
- **NDDS_Transport_Property_t::allow_multicast_interfaces_list** (p. 2126),
- **NDDS_Transport_Property_t::deny_multicast_interfaces_list** (p. 2127)

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

<code>property_in</code>	<< <i>in</i> >> (p. 1013) Desired behavior of this transport. May be NULL for default property.

Returns

A UDPv6 Transport Plugin instance on success; or NULL on failure.

5.115.4.2 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 Connex. Multiple instances of the plugin may be useful for partitioning the network interfaces across multiple RTI Connex 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. 2155):

- **NDDS_Transport_Property_t::allow_interfaces_list** (p. 2125),
- **NDDS_Transport_Property_t::deny_interfaces_list** (p. 2126),
- **NDDS_Transport_Property_t::allow_multicast_interfaces_list** (p. 2126),
- **NDDS_Transport_Property_t::deny_multicast_interfaces_list** (p. 2127)

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

<i>default_network_address_out</i>	<< <i>out</i> >> (p. 1013) Network address to be used when registering the transport.
<i>property_in</i>	<< <i>in</i> >> (p. 1013) Desired behavior of this transport as defined in the DDS_DomainParticipantQos::property (p. 1730).

Returns

A UDPv6 Transport Plugin instance on success; or NULL on failure.

5.115.4.3 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

<i>default_network_address_out</i>	<< out >> (p. 1013) Network address to be used when registering the transport.
<i>property_in</i>	<< in >> (p. 1013) Desired behavior of this transport as defined in the DDS_DomainParticipantQos::property (p. 1730).
<i>propertyPrefix</i>	a prefix for the properties. Expected UDP properties have the form prefix.property_name

5.116 AsyncWaitSet

<<**extension**>> (p. 1012) A specialization of **DDS_WaitSet** (p. 1371) that provides a mechanism to perform the wait asynchronously and uses a thread pool to dispatch the attached active **DDS_Condition** (p. 1370).

Data Structures

- struct **DDS_AsyncWaitSetProperty_t**
Specifies the **DDS_AsyncWaitSet** (p. 1070) behavior.
- struct **DDS_AsyncWaitSetListener**
<<**interface**>> (p. 1013) Listener for receiving event notifications related to the thread pool of the **DDS_AsyncWaitSet** (p. 1070).

Macros

- #define **DDS_AsyncWaitSetListener_INITIALIZER**
Initializer for new **DDS_AsyncWaitSetListener** (p. 1543).

Typedefs

- typedef struct DDS_AsyncWaitSetImpl **DDS_AsyncWaitSet**
A class for dispatching **DDS_Condition** (p. 1370) objects using separate threads of execution. You can see this class as an extension of a **DDS_WaitSet** (p. 1371) that allows asynchronously waiting for the attached **DDS_Condition** (p. 1370) objects to trigger and provide a notification by calling **DDS_Condition_dispatch** (p. 1375).
- typedef struct DDS_AsyncWaitSetCompletionTokenImpl **DDS_AsyncWaitSetCompletionToken**
<<interface>> (p. 1013) Implementation of the completion token role element of the asynchronous completion token pattern that is part of the **DDS_AsyncWaitSet** (p. 1070) behavior.
- typedef void(* **DDS_AsyncWaitSetListener_OnThreadSpawnedCallback**) (void *listener_data, **DDS_↔**
UnsignedLongLong thread_id)
Prototype of a **DDS_AsyncWaitSetListener::on_thread_spawned** (p. 1544) function.
- typedef void(* **DDS_AsyncWaitSetListener_OnThreadDeletedCallback**) (void *listener_data, **DDS_↔**
UnsignedLongLong thread_id)
Prototype of a **DDS_AsyncWaitSetListener::on_thread_deleted** (p. 1544) function.
- typedef void(* **DDS_AsyncWaitSetListener_OnWaitTimeoutCallback**) (void *listener_data, **DDS_↔**
UnsignedLongLong thread_id)
Prototype of a **DDS_AsyncWaitSetListener::on_wait_timeout** (p. 1544) function.
- typedef struct DDS_DataReaderStatusConditionHandlerImpl **DDS_DataReaderStatusConditionHandler**
<<interface>> (p. 1013) Realization of a **DDS_ConditionHandler** (p. 1570) that handles the status of a **DDS_Data↔**
Reader (p. 615).

Functions

- **DDS_ReturnCode_t DDS_AsyncWaitSetCompletionToken_wait** (**DDS_AsyncWaitSetCompletionToken**
*self, const struct **DDS_Duration_t** *max_wait)
Waits for the request associated to an operation made on the **DDS_AsyncWaitSet** (p. 1070) to complete.
- **DDS_ReturnCode_t DDS_AsyncWaitSet_get_property** (**DDS_AsyncWaitSet** *self, struct **DDS_Async↔**
WaitSetProperty_t *property)
Retrieves the **DDS_AsyncWaitSetProperty_t** (p. 1544) configuration of the associated **DDS_AsyncWaitSet** (p. 1070).
- **DDS_ReturnCode_t DDS_AsyncWaitSet_detach_condition** (**DDS_AsyncWaitSet** *self, **DDS_Condition**
*condition)
Detaches the specified **DDS_Condition** (p. 1370) from this **DDS_AsyncWaitSet** (p. 1070).
- **DDS_ReturnCode_t DDS_AsyncWaitSet_detach_condition_with_completion_token** (**DDS_AsyncWait↔**
Set *self, **DDS_Condition** *condition, **DDS_AsyncWaitSetCompletionToken** *completion_token)
Detaches the specified **DDS_Condition** (p. 1370) from this **DDS_AsyncWaitSet** (p. 1070).
- **DDS_ReturnCode_t DDS_AsyncWaitSet_attach_condition** (**DDS_AsyncWaitSet** *self, **DDS_Condition**
*condition)
Attaches the specified **DDS_Condition** (p. 1370) to this **DDS_AsyncWaitSet** (p. 1070).
- **DDS_ReturnCode_t DDS_AsyncWaitSet_attach_condition_with_completion_token** (**DDS_AsyncWaitSet**
*self, **DDS_Condition** *condition, **DDS_AsyncWaitSetCompletionToken** *completion_token)
Attaches the specified **DDS_Condition** (p. 1370) to this **DDS_AsyncWaitSet** (p. 1070).
- **DDS_ReturnCode_t DDS_AsyncWaitSet_unlock_condition** (**DDS_AsyncWaitSet** *self, **DDS_Condition**
*condition)
Allows the **DDS_Condition** (p. 1370) under dispatch to be available for concurrent dispatch from another thread from the pool.
- **DDS_ReturnCode_t DDS_AsyncWaitSet_get_conditions** (**DDS_AsyncWaitSet** *self, struct **DDS_↔**
ConditionSeq *attached_conditions)
Retrieves the list of attached **DDS_Condition** (p. 1370) (s).

- **DDS_ReturnCode_t DDS_AsyncWaitSet_stop (DDS_AsyncWaitSet *self)**
*Initiates the stop procedure on this **DDS_AsyncWaitSet** (p. 1070) that will stop the asynchronous wait.*
- **DDS_ReturnCode_t DDS_AsyncWaitSet_stop_with_completion_token (DDS_AsyncWaitSet *self, DDS_AsyncWaitSetCompletionToken *completion_token)**
*Initiates the stop procedure on this **DDS_AsyncWaitSet** (p. 1070) that will stop the asynchronous wait.*
- **DDS_ReturnCode_t DDS_AsyncWaitSet_start (DDS_AsyncWaitSet *self)**
*Initiates the asynchronous wait on this **DDS_AsyncWaitSet** (p. 1070).*
- **DDS_ReturnCode_t DDS_AsyncWaitSet_start_with_completion_token (DDS_AsyncWaitSet *self, DDS_AsyncWaitSetCompletionToken *completion_token)**
*Initiates the asynchronous wait on this **DDS_AsyncWaitSet** (p. 1070).*
- **DDS_Boolean DDS_AsyncWaitSet_is_started (DDS_AsyncWaitSet *self)**
*Returns whether this **DDS_AsyncWaitSet** (p. 1070) is started.*
- **DDS_AsyncWaitSetCompletionToken * DDS_AsyncWaitSet_create_completion_token (DDS_AsyncWaitSet *self)**
*Creates a new **DDS_AsyncWaitSetCompletionToken** (p. 1073).*
- **DDS_ReturnCode_t DDS_AsyncWaitSet_delete_completion_token (DDS_AsyncWaitSet *self, DDS_AsyncWaitSetCompletionToken *completion_token)**
*Deletes a **DDS_AsyncWaitSetCompletionToken** (p. 1073) previously created from this **DDS_AsyncWaitSet** (p. 1070).*
- **DDS_AsyncWaitSet * DDS_AsyncWaitSet_new (const struct DDS_AsyncWaitSetProperty_t *property)**
*Single-argument constructor that allows creating a **DDS_AsyncWaitSet** (p. 1070) with custom behavior.*
- **DDS_AsyncWaitSet * DDS_AsyncWaitSet_new_with_listener (const struct DDS_AsyncWaitSetProperty_t *property, struct DDS_AsyncWaitSetListener *listener)**
*Constructor that allows specifying a **DDS_AsyncWaitSetListener** (p. 1543).*
- **DDS_AsyncWaitSet * DDS_AsyncWaitSet_new_with_thread_factory (const struct DDS_AsyncWaitSetProperty_t *property, struct DDS_AsyncWaitSetListener *listener, struct DDS_ThreadFactory *thread_factory)**
*Constructor with arguments that allow specifying behavior different than the default one, including specifying a **DDS_ThreadFactory** (p. 2027) for the creation and deletion of the threads within the thread pool.*
- **DDS_ReturnCode_t DDS_AsyncWaitSet_delete (DDS_AsyncWaitSet *self)**
*Deletes a **DDS_AsyncWaitSet** (p. 1070).*
- **DDS_DataReaderStatusConditionHandler * DDS_DataReaderStatusConditionHandler_new (DDS_DataReader *reader, const struct DDS_DataReaderListener *listener, DDS_StatusMask listener_mask)**
*Creates a new **DDS_DataReaderStatusConditionHandler** (p. 1075) instance.*
- **DDS_ReturnCode_t DDS_DataReaderStatusConditionHandler_delete (DDS_DataReaderStatusConditionHandler *self)**
*Deletes a **DDS_DataReaderStatusConditionHandler** (p. 1075) instance previously created with **DDS_DataReaderStatusConditionHandler_new** (p. 1089).*

Variables

- **DDS_AsyncWaitSetCompletionToken *const DDS_ASYNC_WAITSET_COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT**
*For the operations that allow an **DDS_AsyncWaitSetCompletionToken** (p. 1073), this sentinel can be provided to indicate an **DDS_AsyncWaitSet** (p. 1070) to use the implicit completion token and wait on it for request completion.*
- **DDS_AsyncWaitSetCompletionToken *const DDS_ASYNC_WAITSET_COMPLETION_TOKEN_IGNORE**
*For the operations that allow an **DDS_AsyncWaitSetCompletionToken** (p. 1073), this sentinel can be provided to indicate an **DDS_AsyncWaitSet** (p. 1070) to perform the action associating a 'null' completion token.*
- **const struct DDS_AsyncWaitSetProperty_t DDS_ASYNC_WAITSET_PROPERTY_DEFAULT**
*Constant that defines the default property for a **DDS_AsyncWaitSet** (p. 1070).*

5.116.1 Detailed Description

<<*extension*>> (p. 1012) A specialization of **DDS_WaitSet** (p. 1371) that provides a mechanism to perform the wait asynchronously and uses a thread pool to dispatch the attached active **DDS_Condition** (p. 1370).

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.

5.116.2 Macro Definition Documentation

5.116.2.1 DDS_AsyncWaitSetListener_INITIALIZER

```
#define DDS_AsyncWaitSetListener_INITIALIZER
```

Initializer for new **DDS_AsyncWaitSetListener** (p. 1543).

No memory is allocated. New **DDS_AsyncWaitSetListener** (p. 1543) Instances stored in the stack should be initialized with this value before they are passed to any functions.

See also

DDS_AsyncWaitSet_new (p. 1086)

DDS_AsyncWaitSetListener (p. 1543)

5.116.3 Typedef Documentation

5.116.3.1 DDS_AsyncWaitSet

```
typedef struct DDS_AsyncWaitSetImpl DDS_AsyncWaitSet
```

A class for dispatching **DDS_Condition** (p. 1370) objects using separate threads of execution. You can see this class as an extension of a **DDS_WaitSet** (p. 1371) that allows asynchronously waiting for the attached **DDS_Condition** (p. 1370) objects to trigger and provide a notification by calling **DDS_Condition_dispatch** (p. 1375).

DDS_AsyncWaitSet (p. 1070) provides a proactive model to process application events through **DDS_Condition** (p. 1370) objects. **DDS_AsyncWaitSet** (p. 1070) owns a pool of threads to asynchronously wait for the attached **DDS_Condition** (p. 1370) objects to trigger and dispatch them upon wakeup. The asynchronous behavior is the main key different with regards to the **DDS_WaitSet** (p. 1371).

The class diagram and its collaborators is shown below:

5.116.4 AsyncWaitSet Thread Orchestration

DDS_AsyncWaitSet (p. 1070) internally applies a leader-follower pattern for the orchestration of the thread pool. Once a **DDS_AsyncWaitSet** (p. 1070) 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 **DDS_Condition** (p. 1370) to trigger. Remaining threads in the pool, if any, are either idle awaiting to become the leader or busy while processing active **DDS_Condition** (p. 1370).
- Upon wait wakeup, the `Leader` thread resigns its leader status to become a `Processor` thread and dispatch the next active **DDS_Condition** (p. 1370) through the **DDS_Condition_dispatch** (p. 1375) operation.
- One of the `Follower` threads wakes up and becomes the new leader to resume the wait for **DDS_Condition** (p. 1370).

"Thread orchestration in a `::DDS_AsyncWaitSet`"

This behavior implies the following considerations:

- Only one thread a time can wait for the attached **DDS_Condition** (p. 1370) objects to trigger. From a pool of M threads, only one is the leader, P are processing active **DDS_Condition** (p. 1370), and F are idle followers.
- A thread can dispatch only one active **DDS_Condition** (p. 1370) at a time.
- **DDS_AsyncWaitSet** (p. 1070) efficiently distributes threads to dispatch **DDS_Condition** (p. 1370) objects on demand. This avoids underutilizing a thread if **DDS_Condition** (p. 1370) 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 **DDS_AsyncWaitSet** (p. 1070) is not able to wait for more **DDS_Condition** (p. 1370) objects until one thread becomes the leader.

DDS_AsyncWaitSet (p. 1070) has a built-in dispatcher that guarantees fairness and avoids starvation of **DDS_Condition** (p. 1370) objects. By applying a round-robin distribution policy, each attached and active **DDS_Condition** (p. 1370) is dispatched within a finite period of time, assuming the **DDS_ConditionHandler** (p. 1570) always return control after the **DDS_Condition_dispatch** (p. 1375) operation.

5.116.5 AsyncWaitSet Thread Safety

A key aspect of the **DDS_AsyncWaitSet** (p. 1070) is the thread safety. **DDS_AsyncWaitSet** (p. 1070) interface is thread safe, so you can concurrently call any operation on the **DDS_AsyncWaitSet** (p. 1070) object from multiple threads in your application.

Furthermore, **DDS_AsyncWaitSet** (p. 1070) also safely interacts with its own thread pool. Internally, the **DDS_AsyncWaitSet** (p. 1070) applies the asynchronous completion token pattern to perform activities that involve synchronization with the thread pool.

For instance to detach a **DDS_Condition** (p. 1370), the **DDS_AsyncWaitSet** (p. 1070) 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 **DDS_AsyncWaitSet** (p. 1070) is started. Otherwise the internal request will be directly executed by the calling thread.

For a finer control on this behavior, each **DDS_AsyncWaitSet** (p. 1070) 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 **DDS_AsyncWaitSetCompletionToken** (p. 1073). The **DDS_AsyncWaitSet** (p. 1070) creates and reuses **DDS_AsyncWaitSetCompletionToken** (p. 1073) 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 **DDS_AsyncWaitSetCompletionToken** (p. 1073) 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 **DDS_AsyncWaitSet** (p. 1070).

5.116.5.1 Condition Locking

DDS_AsyncWaitSet (p. 1070) incorporates a safety mechanism that prevents calling **DDS_Condition_dispatch** (p. 1375) concurrently. **DDS_AsyncWaitSet** (p. 1070) locks the **DDS_Condition** (p. 1370) 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 **DDS_Condition** (p. 1370) 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 **DDS_AsyncWaitSet** (p. 1070) to enter in a continuous 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. **DDS_AsyncWaitSet** (p. 1070) will still allow you to unlock a **DDS_Condition** (p. 1370) so any other available thread can dispatch the same condition concurrently while preventing the above mentioned problems. You can achieve this by calling **DDS_AsyncWaitSet_unlock_condition** (p. 1080) 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.

5.116.6 AsyncWaitSet Events and Resources

Besides **DDS_Condition** (p. 1370) processing, you can listen to other kind of internal events related to the **DDS_AsyncWaitSet** (p. 1070) and its thread pool by means of the **DDS_AsyncWaitSetListener** (p. 1543).

DDS_AsyncWaitSet (p. 1070) exposes operations to start and stop the asynchronous wait, which involves the creation and deletion of the thread pool respectively.

DDS_AsyncWaitSet (p. 1070) relies on thread-specific storage to provide the described functionality. Each application thread that calls an operation on a **DDS_AsyncWaitSet** (p. 1070) will generate resources that will be associated with such thread. You can free these resources upon thread termination by calling **DDS_DomainParticipantFactory_unregister_thread** (p. 58).

MT Safety:

Safe.

See also

DDS_WaitSet (p. 1371)

DDS_Condition (p. 1370)

DDS_AsyncWaitSetListener (p. 1543)

DDS_AsyncWaitSetCompletionToken (p. 1073).

DDS_AsyncWaitSetProperty_t (p. 1544)

5.116.6.1 DDS_AsyncWaitSetCompletionToken

```
typedef struct DDS_AsyncWaitSetCompletionTokenImpl DDS_AsyncWaitSetCompletionToken
```

<<**interface**>> (p. 1013) Implementation of the completion token role element of the asynchronous completion token pattern that is part of the **DDS_AsyncWaitSet** (p. 1070) behavior.

A **DDS_AsyncWaitSetCompletionToken** (p. 1073) can be in one of the following states:

- **READY**: The completion token can be used to associate a new request. Calling **DDS_AsyncWaitSetCompletionToken_wait** (p. 1076) 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 **DDS_AsyncWaitSetCompletionToken_wait** (p. 1076) 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 **DDS_AsyncWaitSetCompletionToken_wait** (p. 1076) yet. Calling **DDS_AsyncWaitSetCompletionToken_wait** (p. 1076) 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.

5.116.7 AsyncWaitSetCompletionToken management

The same **DDS_AsyncWaitSetCompletionToken** (p. 1073) instance can be 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 **DDS_AsyncWaitSet** (p. 1070) operation that associates the completion token will fail with **DDS_RETURNCODE_PRECONDITION_NOT_MET** (p. 1222).

The completion token functionality can be viewed as a **DDS_AsyncWaitSet** (p. 1070) internal detail from which your application should not need to know. In general, it is recommended to use the default flavor of **DDS_AsyncWaitSet** (p. 1070) 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 **DDS_AsyncWaitSet** (p. 1070).

MT Safety:

Safe.

See also

DDS_AsyncWaitSet (p. 1070)

5.116.7.1 DDS_AsyncWaitSetListener_OnThreadSpawnedCallback

```
typedef void(* DDS_AsyncWaitSetListener_OnThreadSpawnedCallback) (void *listener_data,  DDS_↵
UnsignedLongLong thread_id)
```

Prototype of a **DDS_AsyncWaitSetListener::on_thread_spawned** (p. 1544) function.

Each thread that comprises the thread pool of the **DDS_AsyncWaitSet** (p. 1070) will invoke this operation sequentially as soon as the thread is spawned and right before it becomes the leader or a follower thread.

This callback is invoked by each thread comprising the pool and from its own context, right after it is spawned and the underlying operating system has allocated the necessary resources.

Parameters

<i>listener_data</i>	<< in >> (p. 1013) Data associated with the listener when the listener is set.
<i>thread_id</i>	<< in >> (p. 1013) Thread ID of the current context.

5.116.7.2 DDS_AsyncWaitSetListener_OnThreadDeletedCallback

```
typedef void(* DDS_AsyncWaitSetListener_OnThreadDeletedCallback) (void *listener_data,  DDS_↵
UnsignedLongLong thread_id)
```

Prototype of a **DDS_AsyncWaitSetListener::on_thread_deleted** (p. 1544) function.

Each thread that comprises the thread pool of the **DDS_AsyncWaitSet** (p. 1070) will invoke this operation sequentially right before the thread finalizes its execution due to a stop request or an internal error.

This callback is invoked by each thread comprising the pool and from its own context, right before the underlying operating system releases the associated resources.

Parameters

<i>listener_data</i>	<< in >> (p. 1013) Data associated with the listener when the listener is set.
<i>thread_id</i>	<< in >> (p. 1013) Thread ID of the current context.

5.116.7.3 DDS_AsyncWaitSetListener_OnWaitTimeoutCallback

```
typedef void(* DDS_AsyncWaitSetListener_OnWaitTimeoutCallback) (void *listener_data, DDS_UnsignedLongLong thread_id)
```

Prototype of a **DDS_AsyncWaitSetListener::on_wait_timeout** (p. 1544) function.

The leader thread of the **DDS_AsyncWaitSet** (p. 1070) invokes this callback each time the wait operation timed out while waiting for the attached **DDS_Condition** (p. 1370) objects to trigger.

Parameters

<i>listener_data</i>	<< <i>in</i> >> (p. 1013) Data associated with the listener when the listener is set.
<i>thread_id</i>	<< <i>in</i> >> (p. 1013) Thread ID of the current context. It always corresponds to the current leader thread.

See also

DDS_WaitSet_wait (p. 1382)

5.116.7.4 DDS_DataReaderStatusConditionHandler

```
typedef struct DDS_DataReaderStatusConditionHandlerImpl DDS_DataReaderStatusConditionHandler
```

<<*interface*>> (p. 1013) Realization of a **DDS_ConditionHandler** (p. 1570) that handles the status of a **DDS_DataReader** (p. 615).

A **DDS_DataReaderStatusConditionHandler** (p. 1075) demultiplexes a **DDS_DataReader** (p. 615) status change into the corresponding callback of a provided **DDS_DataReaderListener** (p. 1593) implementation.

Note that the **DDS_DataReaderListener** (p. 1593) notifications have different considerations than if the were made by the **DDS_DataReader** (p. 615) directly:

- Context: The **DDS_DataReaderListener** (p. 1593) callback context is the one of the thread that dispatches the **DDS_Condition** (p. 1370) where this handler is set. For instance, if you attach the condition to a **DDS_AsyncWaitSet** (p. 1070), the context will be one of the threads within the pool.
- Status clearing: All the **DDS_DataReader** (p. 615)'s enabled statuses are cleared upon condition dispatch except the **DDS_DATA_AVAILABLE_STATUS** (p. 1230), which will not be cleared until your application reads the data.
- Exclusive Area: Restrictions depend on the context of the dispatching thread. For instance, if the **DDS_AsyncWaitSet** (p. 1070) dispatches the condition, the listener notifications are free of any exclusive area restrictions.

The **DDS_DataReaderStatusConditionHandler** (p. 1075) is a convenience to handle the status changes of a **DDS_DataReader** (p. 615). You can install a **DDS_DataReaderStatusConditionHandler** (p. 1075) as the handler of a reader's **DDS_StatusCondition** (p. 1371). You can then attach it to a **DDS_WaitSet** (p. 1371) or **DDS_AsyncWaitSet** (p. 1070) and receive status changes notifications through a specific **DDS_DataReaderListener** (p. 1593) implementation instance.

5.116.8 Function Documentation

5.116.8.1 DDS_AsyncWaitSetCompletionToken_wait()

```
DDS_ReturnCode_t DDS_AsyncWaitSetCompletionToken_wait (
    DDS_AsyncWaitSetCompletionToken * self,
    const struct DDS_Duration_t * max_wait )
```

Waits for the request associated to an operation made on the **DDS_AsyncWaitSet** (p. 1070) to complete.

This operation will block the calling thread for a maximum amount of time specified by `max_wait` until the **DDS_AsyncWaitSet** (p. 1070) 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 **DDS_AsyncWaitSet** (p. 1070), it will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

If the operation failed with **DDS_RETCODE_TIMEOUT** (p. 1222) your application can wait again on this completion token.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>max_wait</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL. Maximum time to wait for the request associated to this completion token to complete before timeout.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_AsyncWaitSet (p. 1070)

5.116.8.2 DDS_AsyncWaitSet_get_property()

```
DDS_ReturnCode_t DDS_AsyncWaitSet_get_property (
    DDS_AsyncWaitSet * self,
    struct DDS_AsyncWaitSetProperty_t * property )
```

Retrieves the **DDS_AsyncWaitSetProperty_t** (p. 1544) configuration of the associated **DDS_AsyncWaitSet** (p. 1070).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>property</i>	<< <i>out</i> >> (p. 1013)

Returns

One of the **Standard Return Codes** (p. 1221)

5.116.8.3 DDS_AsyncWaitSet_detach_condition()

```
DDS_ReturnCode_t DDS_AsyncWaitSet_detach_condition (
    DDS_AsyncWaitSet * self,
    DDS_Condition * condition )
```

Detaches the specified **DDS_Condition** (p. 1370) from this **DDS_AsyncWaitSet** (p. 1070).

This operation is equivalent to call **DDS_AsyncWaitSet_detach_condition_with_completion_token** (p. 1077) providing **DDS_ASYNC_WAITSET_COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT** (p. 1090) as a *completion_token*.

This operation blocks until the detach request completes. Upon successful return, it is guaranteed that the specified **DDS_Condition** (p. 1370) is detached.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>condition</i>	<< <i>in</i> >> (p. 1013) DDS_Condition (p. 1370) to be detached.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_AsyncWaitSet_detach_condition_with_completion_token (p. 1077)

5.116.8.4 DDS_AsyncWaitSet_detach_condition_with_completion_token()

```
DDS_ReturnCode_t DDS_AsyncWaitSet_detach_condition_with_completion_token (
    DDS_AsyncWaitSet * self,
```

```

DDS_Condition * condition,
DDS_AsyncWaitSetCompletionToken * completion_token )

```

Detaches the specified **DDS_Condition** (p. 1370) from this **DDS_AsyncWaitSet** (p. 1070).

If this operation succeeds, a detach request has been scheduled and your application can use the provided `completion_token` to wait for this **DDS_AsyncWaitSet** (p. 1070) to process the request. If the **DDS_AsyncWaitSetCompletionToken_wait** (p. 1076) operation returns successfully, it is guaranteed that the **DDS_Condition** (p. 1370) is detached from this **DDS_AsyncWaitSet** (p. 1070).

Once the **DDS_Condition** (p. 1370) is detached, it is guaranteed that the **DDS_AsyncWaitSet** (p. 1070) will no longer process it so it is safe for your application to release any resources associated with the detached **DDS_Condition** (p. 1370).

DDS_Condition (p. 1370) may be detached at any time independently of the state of the **DDS_AsyncWaitSet** (p. 1070).

If the same **DDS_Condition** (p. 1370) had been previously attached multiple times, only a single detach is required; the **DDS_Condition** (p. 1370) is considered attached only once.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>condition</i>	<< <i>in</i> >> (p. 1013) DDS_Condition (p. 1370) to be detached.
<i>completion_token</i>	<< <i>inout</i> >> (p. 1013) a valid DDS_AsyncWaitSetCompletionToken (p. 1073) instance that can be used by your application to wait for the detach request to complete. You can provide one of the special sentinels DDS_ASYNC_WAITSET_COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT (p. 1090) and DDS_ASYNC_WAITSET_COMPLETION_TOKEN_IGNORE (p. 1090).

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_AsyncWaitSet_detach_condition (p. 1077)

DDS_AsyncWaitSet_attach_condition_with_completion_token (p. 1079)

DDS_AsyncWaitSetCompletionToken_wait (p. 1076)

5.116.8.5 DDS_AsyncWaitSet_attach_condition()

```

DDS_ReturnCode_t DDS_AsyncWaitSet_attach_condition (
    DDS_AsyncWaitSet * self,
    DDS_Condition * condition )

```

Attaches the specified **DDS_Condition** (p. 1370) to this **DDS_AsyncWaitSet** (p. 1070).

This operation is equivalent to calling **DDS_AsyncWaitSet_attach_condition_with_completion_token** (p. 1079) providing **DDS_ASYNC_WAITSET_COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT** (p. 1090) as a `completion_token`.

This operation will block until the attach request completes. Upon successful return, it is guaranteed that the specified **DDS_Condition** (p. 1370) is attached.

If the same **DDS_Condition** (p. 1370) is attached multiple times, only the first attachment has any effect. Subsequent attempts to attach the same **DDS_Condition** (p. 1370) are silently ignored. As a result, the **DDS_Condition** (p. 1370) is only considered attached once and must only be detached once.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>condition</i>	<< <i>in</i> >> (p. 1013) DDS_Condition (p. 1370) to be attached.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_AsyncWaitSet_attach_condition_with_completion_token (p. 1079)

5.116.8.6 DDS_AsyncWaitSet_attach_condition_with_completion_token()

```
DDS_ReturnCode_t DDS_AsyncWaitSet_attach_condition_with_completion_token (
    DDS_AsyncWaitSet * self,
    DDS_Condition * condition,
    DDS_AsyncWaitSetCompletionToken * completion_token )
```

Attaches the specified **DDS_Condition** (p. 1370) to this **DDS_AsyncWaitSet** (p. 1070).

If this operation succeeds, an attach request has been scheduled and your application can use the output parameter `completion_token` to wait for this **DDS_AsyncWaitSet** (p. 1070) to process the request. **DDS_AsyncWaitSetCompletionToken_wait** (p. 1076) operation returns successfully, it is guaranteed that the **DDS_Condition** (p. 1370) is attached to this **DDS_AsyncWaitSet** (p. 1070).

Once the **DDS_Condition** (p. 1370) is attached, its trigger value may cause the leader thread of the **DDS_AsyncWaitSet** (p. 1070) to wake up call the **DDS_Condition_dispatch** (p. 1375) operation.

DDS_Condition (p. 1370) may be attached at any time independently of the state of the **DDS_AsyncWaitSet** (p. 1070).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>condition</i>	<< <i>in</i> >> (p. 1013) DDS_Condition (p. 1370) to be attached.
<i>completion_token</i> Generated by Doxygen	<< <i>inout</i> >> (p. 1013) a valid DDS_AsyncWaitSetCompletionToken (p. 1073) instance that can be used by your application to wait for the attach request to complete. You can provide one of the special sentinels DDS_ASYNC_WAITSET_COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT (p. 1090) and DDS_ASYNC_WAITSET_COMPLETION_TOKEN_IGNORE (p. 1090).

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_AsyncWaitSet_attach_condition (p. 1078)

DDS_AsyncWaitSet_detach_condition_with_completion_token (p. 1077)

DDS_AsyncWaitSetCompletionToken_wait (p. 1076)

5.116.8.7 DDS_AsyncWaitSet_unlock_condition()

```
DDS_ReturnCode_t DDS_AsyncWaitSet_unlock_condition (
    DDS_AsyncWaitSet * self,
    DDS_Condition * condition )
```

Allows the **DDS_Condition** (p. 1370) 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 **DDS_Condition** (p. 1370) this **DDS_AsyncWaitSet** (p. 1070) is dispatching. After successfully calling this operation, if the **DDS_Condition** (p. 1370) becomes active this **DDS_AsyncWaitSet** (p. 1070) is allowed to dispatch it again from any available thread from the pool.

You may call this operation any time you need the same **DDS_Condition** (p. 1370) to be dispatched concurrently.

This operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) if you call it from a different context than the dispatch callback or on a different **DDS_Condition** (p. 1370).

Returns

One of the **Standard Return Codes** (p. 1221)

5.116.8.8 DDS_AsyncWaitSet_get_conditions()

```
DDS_ReturnCode_t DDS_AsyncWaitSet_get_conditions (
    DDS_AsyncWaitSet * self,
    struct DDS_ConditionSeq * attached_conditions )
```

Retrieves the list of attached **DDS_Condition** (p. 1370) (s).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>attached_conditions</i>	<< <i>inout</i> >> (p. 1013) a DDS_ConditionSeq (p. 1571) object where the list of attached conditions will be returned.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_AsyncWaitSet_attach_condition (p. 1078)

DDS_AsyncWaitSet_detach_condition (p. 1077)

5.116.8.9 DDS_AsyncWaitSet_stop()

```
DDS_ReturnCode_t DDS_AsyncWaitSet_stop (
    DDS_AsyncWaitSet * self )
```

Initiates the stop procedure on this **DDS_AsyncWaitSet** (p. 1070) that will stop the asynchronous wait.

This operation is equivalent to calling **DDS_AsyncWaitSet_stop_with_completion_token** (p. 1081) providing **DDS_↵_ASYNC_WAITSET_COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT** (p. 1090) as a *completion_token*.

This operation will block until the stop request completes. Upon successful return, it is guaranteed that this **DDS_↵_AsyncWaitSet** (p. 1070) stopped the asynchronous wait and dispatch.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_AsyncWaitSet_start_with_completion_token (p. 1082)

DDS_AsyncWaitSet_stop (p. 1081)

5.116.8.10 DDS_AsyncWaitSet_stop_with_completion_token()

```
DDS_ReturnCode_t DDS_AsyncWaitSet_stop_with_completion_token (
    DDS_AsyncWaitSet * self,
    DDS_AsyncWaitSetCompletionToken * completion_token )
```

Initiates the stop procedure on this **DDS_AsyncWaitSet** (p. 1070) 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 **DDS_AsyncWaitSet** (p. 1070) to process the request. If the **DDS_↵_AsyncWaitSetCompletionToken_wait** (p. 1076) operation returns successfully, it is guaranteed that the thread pool has been deleted and this **DDS_AsyncWaitSet** (p. 1070) no longer process any of the attached **DDS_Condition** (p. 1370) objects.

Once this **DDS_AsyncWaitSet** (p. 1070) is stopped, the **DDS_Condition_dispatch** (p. 1375) will no longer be called on any of the attached **DDS_Condition** (p. 1370), no matter what their trigger value is.

The stop procedure causes the **DDS_AsyncWaitSet** (p. 1070) 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 **DDS_↵_AsyncWaitSetListener** (p. 1543) is installed, this **DDS_AsyncWaitSet** (p. 1070) will sequentially invoke the **DDS_↵_AsyncWaitSetListener::on_thread_deleted** (p. 1544) once per deleted thread.

If this **DDS_AsyncWaitSet** (p. 1070) is already stopped, this operation will return immediately with success, and waiting on the *completion_token* will also return immediately with success.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>completion_token</i>	<< <i>inout</i> >> (p. 1013) a valid DDS_AsyncWaitSetCompletionToken (p. 1073) instance that can be used by your application to wait for the stop request to complete. You can provide one of the special sentinels DDS_ASYNC_WAITSET_COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT (p. 1090) and DDS_ASYNC_WAITSET_COMPLETION_TOKEN_IGNORE (p. 1090).

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_AsyncWaitSet_stop (p. 1081)

DDS_AsyncWaitSet_start_with_completion_token (p. 1082)

5.116.8.11 DDS_AsyncWaitSet_start()

```
DDS_ReturnCode_t DDS_AsyncWaitSet_start (
    DDS_AsyncWaitSet * self )
```

Initiates the asynchronous wait on this **DDS_AsyncWaitSet** (p. 1070).

This operation is equivalent to calling **DDS_AsyncWaitSet_start_with_completion_token** (p. 1082) providing **DDS_↵_ASYNC_WAITSET_COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT** (p. 1090) as a *completion_token*.

This operation blocks until the start request completes. Upon successful return, it is guaranteed that this **DDS_Async_↵WaitSet** (p. 1070) has initiated the asynchronous wait and dispatch.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_AsyncWaitSet_start_with_completion_token (p. 1082)

DDS_AsyncWaitSet_stop (p. 1081)

5.116.8.12 DDS_AsyncWaitSet_start_with_completion_token()

```
DDS_ReturnCode_t DDS_AsyncWaitSet_start_with_completion_token (
    DDS_AsyncWaitSet * self,
    DDS_AsyncWaitSetCompletionToken * completion_token )
```

Initiates the asynchronous wait on this **DDS_AsyncWaitSet** (p. 1070).

If this operation succeeds, a start request has been scheduled and your application can use the provided `completion_token` to wait for this **DDS_AsyncWaitSet** (p. 1070) to process the request. If the **DDS_AsyncWaitSetCompletionToken_wait** (p. 1076) operation returns successfully, it is guaranteed that the thread pool has been created and the leader thread is waiting for the attached **DDS_Condition** (p. 1370) to trigger.

Once this **DDS_AsyncWaitSet** (p. 1070) is started, attached **DDS_Condition** (p. 1370) will be dispatched through the **DDS_Condition_dispatch** (p. 1375) operation when they trigger.

The start procedure causes the **DDS_AsyncWaitSet** (p. 1070) 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 **DDS_AsyncWaitSetListener** (p. 1543) is installed, this **DDS_AsyncWaitSet** (p. 1070) will sequentially invoke the **DDS_AsyncWaitSetListener::on_thread_spawned** (p. 1544) once per spawned thread.

A **DDS_AsyncWaitSet** (p. 1070) can be restarted after a stop. If this **DDS_AsyncWaitSet** (p. 1070) is already started, this operation will return immediately with success, and waiting on the `completion_token` will also return immediately with success.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>completion_token</i>	<< <i>inout</i> >> (p. 1013) a valid DDS_AsyncWaitSetCompletionToken (p. 1073) instance that can be used by your application to wait for the start request to complete. You can provide one of the special sentinels DDS_ASYNC_WAITSET_COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT (p. 1090) and DDS_ASYNC_WAITSET_COMPLETION_TOKEN_IGNORE (p. 1090).

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_AsyncWaitSet_start (p. 1082)

DDS_AsyncWaitSet_stop_with_completion_token (p. 1081)

DDS_WaitSet_wait (p. 1382)

5.116.8.13 DDS_AsyncWaitSet_is_started()

```
DDS_Boolean DDS_AsyncWaitSet_is_started (
    DDS_AsyncWaitSet * self )
```

Returns whether this **DDS_AsyncWaitSet** (p. 1070) is started.

A **DDS_AsyncWaitSet** (p. 1070) is started if all the threads within the thread pool have been created and are running.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if started.

5.116.8.14 DDS_AsyncWaitSet_create_completion_token()

```
DDS_AsyncWaitSetCompletionToken * DDS_AsyncWaitSet_create_completion_token (
    DDS_AsyncWaitSet * self )
```

Creates a new **DDS_AsyncWaitSetCompletionToken** (p. 1073).

All the created **DDS_AsyncWaitSetCompletionToken** (p. 1073) must be deleted by calling **DDS_AsyncWaitSet_delete_completion_token** (p. 1085).

Returns

A new **DDS_AsyncWaitSetCompletionToken** (p. 1073) or NULL on error.

See also

DDS_AsyncWaitSet_delete_completion_token (p. 1085)

DDS_AsyncWaitSet_delete (p. 1088)

5.116.8.15 DDS_AsyncWaitSet_delete_completion_token()

```
DDS_ReturnCode_t DDS_AsyncWaitSet_delete_completion_token (
    DDS_AsyncWaitSet * self,
    DDS_AsyncWaitSetCompletionToken * completion_token )
```

Deletes a **DDS_AsyncWaitSetCompletionToken** (p. 1073) previously created from this **DDS_AsyncWaitSet** (p. 1070).

This operation will fail if the specified `completion_token` was not created from this **DDS_AsyncWaitSet** (p. 1070).

This operation will fail if the specified `completion_token` is associated with a request that has not completed yet.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>completion_token</i>	<< <i>inout</i> >> (p. 1013) a valid DDS_AsyncWaitSetCompletionToken (p. 1073) created from this DDS_AsyncWaitSet (p. 1070).

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_AsyncWaitSet_create_completion_token (p. 1085)

DDS_AsyncWaitSet_delete (p. 1088)

5.116.8.16 **DDS_AsyncWaitSet_new()**

```
DDS_AsyncWaitSet * DDS_AsyncWaitSet_new (
    const struct DDS_AsyncWaitSetProperty_t * property )
```

Single-argument constructor that allows creating a **DDS_AsyncWaitSet** (p. 1070) with custom behavior.

You can provide **DDS_ASYNC_WAITSET_PROPERTY_DEFAULT** (p. 1090) as *property* to create an **DDS_AsyncWaitSet** (p. 1070) with default behavior.

The **DDS_AsyncWaitSet** (p. 1070) is created with no listener installed.

Parameters

<i>property</i>	<< <i>in</i> >> (p. 1013) configuration DDS_AsyncWaitSetProperty_t (p. 1544)

Returns

A new **DDS_AsyncWaitSet** (p. 1070) or NULL if one could not be allocated.

5.116.8.17 **DDS_AsyncWaitSet_new_with_listener()**

```
DDS_AsyncWaitSet * DDS_AsyncWaitSet_new_with_listener (
    const struct DDS_AsyncWaitSetProperty_t * property,
    struct DDS_AsyncWaitSetListener * listener )
```

Constructor that allows specifying a **DDS_AsyncWaitSetListener** (p. 1543).

Creates a new **DDS_AsyncWaitSet** (p. 1070) with the specified property **DDS_AsyncWaitSetProperty_t** (p. 1544) and **DDS_AsyncWaitSetListener** (p. 1543).

Parameters

<i>property</i>	<< <i>in</i> >> (p. 1013) configuration DDS_AsyncWaitSetProperty_t (p. 1544)
<i>listener</i>	<< <i>in</i> >> (p. 1013) the DDS_AsyncWaitSetListener (p. 1543). Cannot be NULL.

Returns

A new **DDS_AsyncWaitSet** (p. 1070) or NULL if one could not be allocated.

5.116.8.18 DDS_AsyncWaitSet_new_with_thread_factory()

```
DDS_AsyncWaitSet * DDS_AsyncWaitSet_new_with_thread_factory (
    const struct DDS_AsyncWaitSetProperty_t * property,
    struct DDS_AsyncWaitSetListener * listener,
    struct DDS_ThreadFactory * thread_factory )
```

Constructor with arguments that allow specifying behavior different than the default one, including specifying a **DDS_↵ThreadFactory** (p. 2027) for the creation and deletion of the threads within the thread pool.

This operation extends **DDS_AsyncWaitSet_new_with_listener** (p. 1086) by allowing to provide a **DDS_Thread_↵Factory** (p. 2027)

Parameters

<i>property</i>	<< <i>in</i> >> (p. 1013) configuration DDS_AsyncWaitSetProperty_t (p. 1544)
<i>listener</i>	<< <i>in</i> >> (p. 1013) the DDS_AsyncWaitSetListener (p. 1543). Cannot be NULL.
<i>thread_factory</i>	<< <i>in</i> >> (p. 1013) DDS_ThreadFactory (p. 2027) for the creation and deletion of threads.

Returns

A new **DDS_AsyncWaitSet** (p. 1070) or NULL if one could not be allocated.

5.116.8.19 DDS_AsyncWaitSet_delete()

```
DDS_ReturnCode_t DDS_AsyncWaitSet_delete (
    DDS_AsyncWaitSet * self )
```

Deletes a **DDS_AsyncWaitSet** (p. 1070).

If the **DDS_AsyncWaitSet** (p. 1070) is started, this operation will initiate the stop procedure and block until it completes.

The deletion will fail if there are outstanding **DDS_AsyncWaitSetCompletionToken** (p. 1073) that have not been deleted.

Any outstanding **DDS_AsyncWaitSet** (p. 1070) must be deleted before finalizing the **DDS_DomainParticipantFactory** (p. 32). Otherwise undefined behavior may occur.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

See also

DDS_AsyncWaitSet_new (p. 1086)

5.116.8.20 DDS_DataReaderStatusConditionHandler_new()

```
DDS_DataReaderStatusConditionHandler * DDS_DataReaderStatusConditionHandler_new (
    DDS_DataReader * reader,
    const struct DDS_DataReaderListener * listener,
    DDS_StatusMask listener_mask )
```

Creates a new **DDS_DataReaderStatusConditionHandler** (p. 1075) instance.

The created DataReaderStatusConditionHandler can set as **DDS_ConditionHandler** (p. 1570) in any **DDS_Condition** (p. 1370) and will demultiplex the specified status changes from the specified **DDS_DataReader** (p. 615)

Parameters

<i>reader</i>	<< <i>in</i> >> (p. 1013) The DDS_DataReader (p. 615) for which the status changes are demultiplexed to the specified <i>listener</i>
<i>listener</i>	<< <i>in</i> >> (p. 1013) that receives the status changes notifications from the specified <i>reader</i> .
<i>listener_mask</i>	<< <i>in</i> >> (p. 1013) Specifies which status changes from the reader to demultiplex to the <i>listener</i> .

Returns

A new instance or NULL on error.

See also

DDS_DataReader (p. 615)

DDS_DataReader_set_listener (p. 690)

DDS_StatusCondition (p. 1371)

DDS_ConditionHandler (p. 1570)

5.116.8.21 DDS_DataReaderStatusConditionHandler_delete()

```
DDS_ReturnCode_t DDS_DataReaderStatusConditionHandler_delete (
    DDS_DataReaderStatusConditionHandler * self )
```

Deletes a **DDS_DataReaderStatusConditionHandler** (p. 1075) instance previously created with **DDS_DataReaderStatusConditionHandler_new** (p. 1089).

Returns

One of the **Standard Return Codes** (p. 1221)

5.116.9 Variable Documentation

5.116.9.1 DDS_ASYNC_WAITSET_COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT

```
DDS_AsyncWaitSetCompletionToken* const DDS_ASYNC_WAITSET_COMPLETION_TOKEN_USE_IMPLICIT_AND_WAIT
```

For the operations that allow an **DDS_AsyncWaitSetCompletionToken** (p. 1073), this sentinel can be provided to indicate an **DDS_AsyncWaitSet** (p. 1070) to use the implicit completion token and wait on it for request completion.

If this sentinel is used, the **DDS_AsyncWaitSet** (p. 1070) will use the completion token associated with the calling thread and will wait with an infinite timeout until the request completes successfully.

5.116.9.2 DDS_ASYNC_WAITSET_COMPLETION_TOKEN_IGNORE

```
DDS_AsyncWaitSetCompletionToken* const DDS_ASYNC_WAITSET_COMPLETION_TOKEN_IGNORE
```

For the operations that allow an **DDS_AsyncWaitSetCompletionToken** (p. 1073), this sentinel can be provided to indicate an **DDS_AsyncWaitSet** (p. 1070) to perform the action associating a 'null' completion token.

This sentinel is a realization of the null object pattern of an **DDS_AsyncWaitSetCompletionToken** (p. 1073). If this object is provided to a **DDS_AsyncWaitSet** (p. 1070) 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 **DDS_AsyncWaitSet** (p. 1070) 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 **DDS_AsyncWaitSet** (p. 1070) without needing to know 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 **DDS_AsyncWaitSet** (p. 1070) from within one of the threads from its thread pool, where waiting on a valid **DDS_AsyncWaitSetCompletionToken** (p. 1073) is forbidden.

5.116.9.3 DDS_ASYNC_WAITSET_PROPERTY_DEFAULT

```
const struct DDS_AsyncWaitSetProperty_t DDS_ASYNC_WAITSET_PROPERTY_DEFAULT
```

Constant that defines the default property for a **DDS_AsyncWaitSet** (p. 1070).

Value: `wait_set_property = DDS_WaitSetProperty_t_INITIALIZER` (p. 1369), `thread_pool_size = 1`, `thread_settings DDS_THREAD_SETTINGS_DEFAULT`, `thread_base_name = NULL`, `wait_timeout = DDS_DURATION_INFINITE` (p. 1208) `level = 1`

5.117 Participant Built-in Topics

Builtin topic for accessing information about the DomainParticipants discovered by RTI Connext.

Data Structures

- struct **DDS_ParticipantBuiltinTopicData**
Entry created when a DomainParticipant object is discovered.
- struct **DDS_ParticipantBuiltinTopicDataSeq**
Instantiates `FooSeq` (p. 2113) < `DDS_ParticipantBuiltinTopicData` (p. 1870) > .
- struct **DDS_ParticipantBuiltinTopicDataTypeSupport**
Instantiates `TypeSupport` < `DDS_ParticipantBuiltinTopicData` (p. 1870) > .

Typedefs

- typedef struct **DDS_ParticipantBuiltinTopicData** **DDS_ParticipantBuiltinTopicData**
Entry created when a DomainParticipant object is discovered.
- typedef struct **DDS_ParticipantBuiltinTopicDataDataReader** **DDS_ParticipantBuiltinTopicDataDataReader**
Instantiates `DataReader` < `DDS_ParticipantBuiltinTopicData` (p. 1870) > .

Variables

- const char * **DDS_PARTICIPANT_TOPIC_NAME**
Participant topic name.

5.117.1 Detailed Description

Builtin topic for accessing information about the DomainParticipants discovered by RTI Connext.

5.117.2 Typedef Documentation

5.117.2.1 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. 1092). It contains QoS policies and additional information that apply to the remote **DDS_DomainParticipant** (p. 77).

See also

DDS_PARTICIPANT_TOPIC_NAME (p. 1092)

DDS_ParticipantBuiltinTopicDataDataReader (p. 1092)

5.117.2.2 DDS_ParticipantBuiltinTopicDataDataReader

```
typedef struct DDS_ParticipantBuiltinTopicDataDataReader DDS_ParticipantBuiltinTopicDataData↵
Reader
```

Instantiates `DataReader < DDS_ParticipantBuiltinTopicData` (p. 1870) > .

DDS_DataReader (p. 615) of topic **DDS_PARTICIPANT_TOPIC_NAME** (p. 1092) used for accessing **DDS_↵**
ParticipantBuiltinTopicData (p. 1870) of the remote **DDS_DomainParticipant** (p. 77).

Instantiates:

<<*generic*>> (p. 1013) **FooDataReader** (p. 2112)

See also

DDS_ParticipantBuiltinTopicData (p. 1870)

DDS_PARTICIPANT_TOPIC_NAME (p. 1092)

5.117.3 Variable Documentation

5.117.3.1 DDS_PARTICIPANT_TOPIC_NAME

```
const char* DDS_PARTICIPANT_TOPIC_NAME [extern]
```

Participant topic name.

Topic name of **DDS_ParticipantBuiltinTopicDataDataReader** (p. 1092)

See also

DDS_ParticipantBuiltinTopicData (p. 1870)

DDS_ParticipantBuiltinTopicDataDataReader (p. 1092)

5.118 Topic Built-in Topics

Data Structures

- struct **DDS_TopicBuiltinTopicData**
Entry created when a Topic object discovered.
- struct **DDS_TopicBuiltinTopicDataSeq**
Instantiates `FooSeq` (p. 2113) < `DDS_TopicBuiltinTopicData` (p. 2034) > .
- struct **DDS_TopicBuiltinTopicDataTypeSupport**
Instantiates `TypeSupport` < `DDS_TopicBuiltinTopicData` (p. 2034) > .

Typedefs

- typedef struct **DDS_TopicBuiltinTopicData DDS_TopicBuiltinTopicData**
Entry created when a Topic object discovered.
- typedef struct **DDS_TopicBuiltinTopicDataDataReader DDS_TopicBuiltinTopicDataDataReader**
Instantiates `DataReader` < `DDS_TopicBuiltinTopicData` (p. 2034) > .

Variables

- const char * **DDS_TOPIC_TOPIC_NAME**
Topic topic name.

5.118.1 Detailed Description

Builtin topic for accessing information about the Topics discovered by RTI Connex.

5.118.2 Typedef Documentation

5.118.2.1 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. 1094). It contains QoS policies and additional information that apply to the remote **DDS_Topic** (p. 175).

Note: The **DDS_TopicBuiltinTopicData** (p. 2034) 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. 1905) and **DDS_SubscriptionBuiltinTopicData** (p. 2012)). Therefore TopicBuiltinTopicData DataReaders will not receive any data.

See also

- DDS_TOPIC_TOPIC_NAME** (p. 1094)
- DDS_TopicBuiltinTopicDataDataReader** (p. 1093)

5.118.2.2 DDS_TopicBuiltinTopicDataDataReader

```
typedef struct DDS_TopicBuiltinTopicDataDataReader DDS_TopicBuiltinTopicDataDataReader
```

Instantiates `DataReader < DDS_TopicBuiltinTopicData (p. 2034) >` .

DDS_DataReader (p. 615) of topic **DDS_TOPIC_TOPIC_NAME** (p. 1094) used for accessing **DDS_TopicBuiltinTopicData** (p. 2034) of the remote **DDS_Topic** (p. 175).

Note: The **DDS_TopicBuiltinTopicData** (p. 2034) 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. 1905) and **DDS_SubscriptionBuiltinTopicData** (p. 2012)). Therefore TopicBuiltinTopicData DataReaders will not receive any data.

Instantiates:

`<<generic>>` (p. 1013) **FooDataReader** (p. 2112)

See also

DDS_TopicBuiltinTopicData (p. 2034)

DDS_TOPIC_TOPIC_NAME (p. 1094)

5.118.3 Variable Documentation

5.118.3.1 DDS_TOPIC_TOPIC_NAME

```
const char* DDS_TOPIC_TOPIC_NAME [extern]
```

Topic topic name.

Topic name of **DDS_TopicBuiltinTopicDataDataReader** (p. 1093)

See also

DDS_TopicBuiltinTopicData (p. 2034)

DDS_TopicBuiltinTopicDataDataReader (p. 1093)

5.119 Publication Built-in Topics

Builtin topic for accessing information about the Publications discovered by RTI Connext.

Data Structures

- struct **DDS_PublicationBuiltinTopicData**
*Entry created when a **DDS_DataWriter** (p. 485) is discovered in association with its Publisher.*
- struct **DDS_PublicationBuiltinTopicDataSeq**
*Instantiates **FooSeq** (p. 2113) < **DDS_PublicationBuiltinTopicData** (p. 1905) > .*
- struct **DDS_PublicationBuiltinTopicDataSupport**
*Instantiates **TypeSupport** < **DDS_PublicationBuiltinTopicData** (p. 1905) > .*

Typedefs

- typedef struct **DDS_PublicationBuiltinTopicData** **DDS_PublicationBuiltinTopicData**
*Entry created when a **DDS_DataWriter** (p. 485) is discovered in association with its Publisher.*
- typedef struct **DDS_PublicationBuiltinTopicDataDataReader** **DDS_PublicationBuiltinTopicDataDataReader**
*Instantiates **DataReader** < **DDS_PublicationBuiltinTopicData** (p. 1905) > .*

Variables

- const char * **DDS_PUBLICATION_TOPIC_NAME**
Publication topic name.

5.119.1 Detailed Description

Builtin topic for accessing information about the Publications discovered by RTI Connext.

5.119.2 Typedef Documentation

5.119.2.1 DDS_PublicationBuiltinTopicData

```
typedef struct DDS_PublicationBuiltinTopicData DDS_PublicationBuiltinTopicData
```

Entry created when a **DDS_DataWriter** (p. 485) is discovered in association with its Publisher.

Data associated with the built-in topic **DDS_PUBLICATION_TOPIC_NAME** (p. 1096). It contains QoS policies and additional information that apply to the remote **DDS_DataWriter** (p. 485) the related **DDS_Publisher** (p. 443).

See also

- DDS_PUBLICATION_TOPIC_NAME** (p. 1096)
- DDS_PublicationBuiltinTopicDataDataReader** (p. 1095)

5.119.2.2 DDS_PublicationBuiltinTopicDataDataReader

```
typedef struct DDS_PublicationBuiltinTopicDataDataReader DDS_PublicationBuiltinTopicDataData↵
Reader
```

Instantiates `DataReader < DDS_PublicationBuiltinTopicData (p. 1905) >` .

DDS_DataReader (p.615) of topic **DDS_PUBLICATION_TOPIC_NAME** (p.1096) used for accessing **DDS_↵**
PublicationBuiltinTopicData (p. 1905) of the remote **DDS_DataWriter** (p.485) and the associated **DDS_Publisher**
(p. 443).

Instantiates:

`<<generic>>` (p. 1013) **FooDataReader** (p. 2112)

See also

DDS_PublicationBuiltinTopicData (p. 1905)

DDS_PUBLICATION_TOPIC_NAME (p. 1096)

5.119.3 Variable Documentation

5.119.3.1 DDS_PUBLICATION_TOPIC_NAME

```
const char* DDS_PUBLICATION_TOPIC_NAME [extern]
```

Publication topic name.

Topic name of **DDS_PublicationBuiltinTopicDataDataReader** (p. 1095)

See also

DDS_PublicationBuiltinTopicData (p. 1905)

DDS_PublicationBuiltinTopicDataDataReader (p. 1095)

5.120 Subscription Built-in Topics

Builtin topic for accessing information about the Subscriptions discovered by RTI Connex.

Data Structures

- struct **DDS_SubscriptionBuiltinTopicData**
*Entry created when a **DDS_DataReader** (p. 615) is discovered in association with its Subscriber.*
- struct **DDS_SubscriptionBuiltinTopicDataSeq**
*Instantiates **FooSeq** (p. 2113) < **DDS_SubscriptionBuiltinTopicData** (p. 2012) > .*
- struct **DDS_SubscriptionBuiltinTopicDataTypeSupport**
*Instantiates **TypeSupport** < **DDS_SubscriptionBuiltinTopicData** (p. 2012) > .*

Typedefs

- typedef struct **DDS_SubscriptionBuiltinTopicData DDS_SubscriptionBuiltinTopicData**
*Entry created when a **DDS_DataReader** (p. 615) is discovered in association with its Subscriber.*
- typedef struct **DDS_SubscriptionBuiltinTopicDataDataReader DDS_SubscriptionBuiltinTopicDataData↵Reader**
*Instantiates **DataReader** < **DDS_SubscriptionBuiltinTopicData** (p. 2012) > .*

Variables

- const char * **DDS_SUBSCRIPTION_TOPIC_NAME**
Subscription topic name.

5.120.1 Detailed Description

Builtin topic for accessing information about the Subscriptions discovered by RTI Connext.

5.120.2 Typedef Documentation

5.120.2.1 DDS_SubscriptionBuiltinTopicData

```
typedef struct DDS_SubscriptionBuiltinTopicData DDS_SubscriptionBuiltinTopicData
```

Entry created when a **DDS_DataReader** (p. 615) is discovered in association with its Subscriber.

Data associated with the built-in topic **DDS_SUBSCRIPTION_TOPIC_NAME** (p. 1098). It contains QoS policies and additional information that apply to the remote **DDS_DataReader** (p. 615) the related **DDS_Subscriber** (p. 572).

See also

- DDS_SUBSCRIPTION_TOPIC_NAME** (p. 1098)
- DDS_SubscriptionBuiltinTopicDataDataReader** (p. 1097)

5.120.2.2 DDS_SubscriptionBuiltinTopicDataDataReader

```
typedef struct DDS_SubscriptionBuiltinTopicDataDataReader DDS_SubscriptionBuiltinTopicDataData↵
Reader
```

Instantiates DataReader < **DDS_SubscriptionBuiltinTopicData** (p. 2012) > .

DDS_DataReader (p. 615) of topic **DDS_SUBSCRIPTION_TOPIC_NAME** (p. 1098) used for accessing **DDS_↵**
SubscriptionBuiltinTopicData (p. 2012) of the remote **DDS_DataReader** (p. 615) and the associated **DDS_↵**
Subscriber (p. 572).

Instantiates:

<<*generic*>> (p. 1013) **FooDataReader** (p. 2112)

See also

DDS_SubscriptionBuiltinTopicData (p. 2012)

DDS_SUBSCRIPTION_TOPIC_NAME (p. 1098)

5.120.3 Variable Documentation

5.120.3.1 DDS_SUBSCRIPTION_TOPIC_NAME

```
const char* DDS_SUBSCRIPTION_TOPIC_NAME [extern]
```

Subscription topic name.

Topic name of **DDS_SubscriptionBuiltinTopicDataDataReader** (p. 1097)

See also

DDS_SubscriptionBuiltinTopicData (p. 2012)

DDS_SubscriptionBuiltinTopicDataDataReader (p. 1097)

5.121 ServiceRequest Built-in Topic

Builtin topic for accessing requests from different services within RTI Connex.

Data Structures

- struct **DDS_ServiceRequest**
A request coming from one of the built-in services.
- struct **DDS_ServiceRequestSeq**
*Instantiates **FooSeq** (p. 2113) < **DDS_ServiceRequest** (p. 2001) > .*
- struct **DDS_ServiceRequestTypeSupport**
*Instantiates **TypeSupport** < **DDS_ServiceRequest** (p. 2001) > .*

Typedefs

- typedef struct **DDS_ServiceRequest** **DDS_ServiceRequest**
A request coming from one of the built-in services.
- typedef struct **DDS_ServiceRequestDataReader** **DDS_ServiceRequestDataReader**
*Instantiates **DataReader** < **DDS_ServiceRequest** (p. 2001) > .*

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 **DDS_TopicQuery** (p. 707) 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.

5.121.1 Detailed Description

Builtin topic for accessing requests from different services within RTI Connext.

Currently, the **DDS_TopicQuery** (p. 707), Locator Reachability Instance State Consistency and Controlability (part of Observability) all rely on this topic.

See also

Topic Queries (p. 703) 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. 706).

5.121.2 Typedef Documentation

5.121.2.1 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. 1101). It contains service-specific information.

See also

DDS_SERVICE_REQUEST_TOPIC_NAME (p. 1101)

DDS_ParticipantBuiltinTopicDataDataReader (p. 1092)

5.121.2.2 DDS_ServiceRequestDataReader

```
typedef struct DDS_ServiceRequestDataReader DDS_ServiceRequestDataReader
```

Instantiates `DataReader < DDS_ServiceRequest (p. 2001) > .`

DDS_DataReader (p. 615) of topic **DDS_SERVICE_REQUEST_TOPIC_NAME** (p. 1101) used for accessing **DDS_↔ServiceRequest** (p. 2001) samples.

Instantiates:

`<<generic>> (p. 1013) FooDataReader` (p. 2112)

See also

DDS_ServiceRequest (p. 2001)

DDS_SERVICE_REQUEST_TOPIC_NAME (p. 1101)

5.121.3 Variable Documentation

5.121.3.1 DDS_UNKNOWN_SERVICE_REQUEST_ID

```
const DDS_Long DDS_UNKNOWN_SERVICE_REQUEST_ID [extern]
```

An invalid Service Id.

5.121.3.2 DDS_TOPIC_QUERY_SERVICE_REQUEST_ID

```
const DDS_Long DDS_TOPIC_QUERY_SERVICE_REQUEST_ID [extern]
```

Service Id for the **DDS_TopicQuery** (p. 707) Service.

5.121.3.3 DDS_LOCATOR_REACHABILITY_SERVICE_REQUEST_ID

```
const DDS_Long DDS_LOCATOR_REACHABILITY_SERVICE_REQUEST_ID [extern]
```

Service Id for the Locator Reachability Service.

5.121.3.4 DDS_INSTANCE_STATE_SERVICE_REQUEST_ID

```
const DDS_Long DDS_INSTANCE_STATE_SERVICE_REQUEST_ID [extern]
```

Service Id for the Instance State Request service.

5.121.3.5 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.

5.121.3.6 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.

5.121.3.7 DDS_SERVICE_REQUEST_TOPIC_NAME

```
const char* DDS_SERVICE_REQUEST_TOPIC_NAME [extern]
```

Service Request topic name.

Topic name of the **DDS_ServiceRequestDataReader** (p. 1100)

See also

DDS_ServiceRequest (p. 2001)

DDS_ServiceRequestDataReader (p. 1100)

5.122 Common types and functions

Types and functions related to the built-in topics.

Data Structures

- struct **DDS_Locator_t**
<<extension>> (p. 795) 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. 1826) >
- struct **DDS_ProtocolVersion_t**
<<extension>> (p. 1012) Type used to represent the version of the RTPS protocol.
- struct **DDS_VendorId_t**
<<extension>> (p. 1012) Type used to represent the vendor of the service implementing the RTPS protocol.
- struct **DDS_ProductVersion_t**
<<extension>> (p. 1012) Type used to represent the current version of RTI Connex.
- struct **DDS_TransportInfo_t**
Contains the class_id and message_size_max of an installed transport.
- struct **DDS_TransportInfoSeq**
Instantiates FooSeq (p. 2113) < DDS_TransportInfo_t (p. 2052) > .
- struct **DDS_BuiltinTopicKey_t**
The key type of the built-in topic types.
- struct **DDS_ContentFilterProperty_t**
<<extension>> (p. 1012) 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.
- **#define DDS_BuiltinTopicKey_t_INITIALIZER**
Initializer for new Keys.

Typedefs

- **typedef struct DDS_Locator_t DDS_Locator_t**
<<extension>> (p. 795) 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. 1012) 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_BuiltinTopicKey_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 accessed via shared memory.
- const **DDS_Long DDS_LOCATOR_KIND_SHMEM_510**
A locator for an address accessed via shared memory for RTI Connex 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 Connex 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 **DDS_Entity** (p. 1360).*
- const struct **DDS_BuiltinTopicKey_t DDS_BUILTIN_TOPIC_KEY_UNKNOWN**
Constant representing an unknown value.

5.122.1 Detailed Description

Types and functions related to the built-in topics.

5.122.2 Macro Definition Documentation

5.122.2.1 DDS_LOCATOR_ADDRESS_LENGTH_MAX

```
#define DDS_LOCATOR_ADDRESS_LENGTH_MAX 16
```

Declares length of address field in locator.

5.122.2.2 DDS_PROTOCOLVERSION_1_0

```
#define DDS_PROTOCOLVERSION_1_0 { 1, 0 }
```

The protocol version 1.0.

5.122.2.3 DDS_PROTOCOLVERSION_1_1

```
#define DDS_PROTOCOLVERSION_1_1 { 1, 1 }
```

The protocol version 1.1.

5.122.2.4 DDS_PROTOCOLVERSION_1_2

```
#define DDS_PROTOCOLVERSION_1_2 { 1, 2 }
```

The protocol version 1.2.

5.122.2.5 DDS_PROTOCOLVERSION_2_0

```
#define DDS_PROTOCOLVERSION_2_0 { 2, 0 }
```

The protocol version 2.0.

5.122.2.6 DDS_PROTOCOLVERSION_2_1

```
#define DDS_PROTOCOLVERSION_2_1 { 2, 1 }
```

The protocol version 2.1.

5.122.2.7 DDS_PROTOCOLVERSION

```
#define DDS_PROTOCOLVERSION { 2, 1 }
```

The most recent protocol version. Currently 2.1.

5.122.2.8 DDS_VENDOR_ID_LENGTH_MAX

```
#define DDS_VENDOR_ID_LENGTH_MAX 2
```

Length of vendor id.

5.122.2.9 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.

5.122.2.10 DDS_BuiltinTopicKey_t_INITIALIZER

```
#define DDS_BuiltinTopicKey_t_INITIALIZER
```

Initializer for new Keys.

5.122.3 Typedef Documentation

5.122.3.1 DDS_Locator_t

```
typedef struct DDS_Locator_t DDS_Locator_t
```

<<**extension**>> (p. 795) Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports.

5.122.3.2 DDS_ProtocolVersion_t

```
typedef struct DDS_ProtocolVersion_t DDS_ProtocolVersion_t
```

<<**extension**>> (p. 1012) Type used to represent the version of the RTPS protocol.

5.122.3.3 DDS_BuiltinTopicKey_t

```
typedef struct DDS_BuiltinTopicKey_t DDS_BuiltinTopicKey_t
```

The key type of the built-in topic types.

Each remote **DDS_Entity** (p. 1360) 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. 1870)

DDS_TopicBuiltinTopicData (p. 2034)

DDS_PublicationBuiltinTopicData (p. 1905)

DDS_SubscriptionBuiltinTopicData (p. 2012)

5.122.4 Function Documentation

5.122.4.1 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

<i>self</i>	<< <i>in</i> >> (p. 1013) This Key. Cannot be NULL.
<i>other</i>	<< <i>in</i> >> (p. 1013) The other Key to be compared with this Key. Cannot be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two Keys have equal values, or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.122.4.2 DDS_BuiltinTopicKey_copy()

```
void DDS_BuiltinTopicKey_copy (
    DDS_BuiltinTopicKey_t * dst,
    const DDS_BuiltinTopicKey_t * src )
```

Copies another Key into this Key.

Parameters

<i>dst</i>	<< out >> (p. 1013) This Key. Cannot be NULL.
<i>src</i>	<< in >> (p. 1013) The other Key to be copied. Cannot be NULL.

5.122.4.3 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

<i>self</i>	<< in >> (p. 1013) This Key. Cannot be NULL.
<i>dst</i>	<< out >> (p. 1013) The destination GUID. Cannot be NULL.

5.122.4.4 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

<i>self</i>	<< out >> (p. 1013) This Key. Cannot be NULL.
<i>src</i>	<< in >> (p. 1013) The GUID to be used to initialize this Key. Cannot be NULL.

5.122.4.5 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

<i>self</i>	<< <i>in</i> >> (p. 1013) This Key. Cannot be NULL.
<i>dst</i>	<< <i>out</i> >> (p. 1013) The destination InstanceHandle. Cannot be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) on success or **DDS_BOOLEAN_FALSE** (p. 1201) on failure

5.122.4.6 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

<i>self</i>	<< <i>out</i> >> (p. 1013) This Key. Cannot be NULL.
<i>src</i>	<< <i>in</i> >> (p. 1013) The InstanceHandle to be used to initialize this Key. Cannot be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) on success or **DDS_BOOLEAN_FALSE** (p. 1201) on failure

5.122.5 Variable Documentation

5.122.5.1 DDS_LOCATOR_INVALID

```
const struct DDS_Locator_t DDS_LOCATOR_INVALID [extern]
```

An invalid locator.

5.122.5.2 DDS_LOCATOR_KIND_INVALID

```
const DDS_Long DDS_LOCATOR_KIND_INVALID [extern]
```

Locator of this kind is invalid.

5.122.5.3 DDS_LOCATOR_PORT_INVALID

```
const DDS_UnsignedLong DDS_LOCATOR_PORT_INVALID [extern]
```

An invalid port.

5.122.5.4 DDS_LOCATOR_ADDRESS_INVALID

```
const DDS_Octet DDS_LOCATOR_ADDRESS_INVALID[ DDS_LOCATOR_ADDRESS_LENGTH_MAX] [extern]
```

An invalid address.

5.122.5.5 DDS_LOCATOR_KIND_UDPv4

```
const DDS_Long DDS_LOCATOR_KIND_UDPv4 [extern]
```

A locator for a UDPv4 address.

5.122.5.6 DDS_LOCATOR_KIND_UDPv4_WAN

```
const DDS_Long DDS_LOCATOR_KIND_UDPv4_WAN [extern]
```

A locator for a UDPv4 asymmetric transport address.

5.122.5.7 DDS_LOCATOR_KIND_SHMEM

```
const DDS_Long DDS_LOCATOR_KIND_SHMEM [extern]
```

A locator for an address accessed via shared memory.

5.122.5.8 DDS_LOCATOR_KIND_SHMEM_510

```
const DDS_Long DDS_LOCATOR_KIND_SHMEM_510 [extern]
```

A locator for an address accessed via shared memory for RTI Connex 5.1.0 and earlier.

5.122.5.9 DDS_LOCATOR_KIND_UDPv6

```
const DDS_Long DDS_LOCATOR_KIND_UDPv6 [extern]
```

A locator for a UDPv6 address.

5.122.5.10 DDS_LOCATOR_KIND_UDPv6_510

```
const DDS_Long DDS_LOCATOR_KIND_UDPv6_510 [extern]
```

A locator for a UDPv6 address for RTI Connex 5.1.0 and earlier.

5.122.5.11 DDS_LOCATOR_KIND_RESERVED

```
const DDS_Long DDS_LOCATOR_KIND_RESERVED [extern]
```

Locator of this kind is reserved.

5.122.5.12 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 **DDS_Entity** (p. 1360).

5.122.5.13 DDS_BUILTIN_TOPIC_KEY_UNKNOWN

```
const struct DDS_BuiltinTopicKey_t DDS_BUILTIN_TOPIC_KEY_UNKNOWN [extern]
```

Constant representing an unknown value.

5.123 String Built-in Type

Built-in type consisting of a single character string.

Data Structures

- struct **DDS_StringTypeSupport**
 <<*interface*>> (p. 1013) *String type support.*

Typedefs

- typedef struct **DDS_StringDataWriter** **DDS_StringDataWriter**
 <<*interface*>> (p. 1013) *Instantiates DataWriter < char* >.*
- typedef struct **DDS_StringDataReader** **DDS_StringDataReader**
 <<*interface*>> (p. 1013) *Instantiates DataReader < char* >.*

Functions

- **DDS_ReturnCode_t DDS_StringTypeSupport_register_type** (**DDS_DomainParticipant** *participant, const char *type_name)
Allows an application to communicate to RTI Connexx the existence of the char data type.*
- **DDS_ReturnCode_t DDS_StringTypeSupport_unregister_type** (**DDS_DomainParticipant** *participant, const char *type_name)
Allows an application to unregister the char data type from RTI Connexx. After calling unregister_type, no further communication using this type is possible.*
- const char * **DDS_StringTypeSupport_get_type_name** (void)
Get the default name for the char type.*
- void **DDS_StringTypeSupport_print_data** (const char *a_data)
 <<*extension*>> (p. 1012) *Print value of data type to standard out.*
- **DDS_TypeCode * DDS_StringTypeSupport_get_typecode** (void)
 <<*extension*>> (p. 1012) *Retrieves the TypeCode for the Type.*
- **DDS_ReturnCode_t DDS_StringTypeSupport_serialize_data_to_cdr_buffer** (char *buffer, unsigned int *length, const char *a_data)
 <<*extension*>> (p. 1012) *Serializes the input sample into a CDR buffer of octets.*
- **DDS_ReturnCode_t DDS_StringTypeSupport_serialize_data_to_cdr_buffer_ex** (char *buffer, unsigned int *length, const char *a_data, **DDS_DataRepresentationId_t** representation)
 <<*extension*>> (p. 1012) *Serializes the input sample into a buffer of octets.*
- **DDS_ReturnCode_t DDS_StringTypeSupport_deserialize_data_from_cdr_buffer** (char **a_data, const char *buffer, unsigned int length)
 <<*extension*>> (p. 1012) *Deserializes a sample from a buffer of octets.*
- **DDS_ReturnCode_t DDS_StringTypeSupport_data_to_string** (const char *sample, char *str, **DDS_UnsignedLong** *str_size, **DDS_PrintFormatProperty** *property)
 <<*extension*>> (p. 1012) *Get the string representation of an input sample.*
- **DDS_StringDataWriter * DDS_StringDataWriter_narrow** (**DDS_DataWriter** *writer)
Narrow the given DDS_DataWriter (p. 485) pointer to a DDS_StringDataWriter (p. 1114) pointer.
- **DDS_DataWriter * DDS_StringDataWriter_as_datawriter** (**DDS_StringDataWriter** *writer)
Widen the given DDS_StringDataWriter (p. 1114) pointer to a DDS_DataWriter (p. 485) pointer.
- char * **DDS_StringDataWriter_create_data** (**DDS_StringDataWriter** *self)
Creates a string data instance.
- **DDS_Boolean DDS_StringDataWriter_delete_data** (**DDS_StringDataWriter** *self, char *sample)
Destroys a string data instance created by DDS_StringDataWriter_create_data (p. 1119).

- **DDS_ReturnCode_t DDS_StringDataWriter_write** (**DDS_StringDataWriter** *self, const char *instance_data, const **DDS_InstanceHandle_t** *handle)
Modifies the value of a string data instance.
- **DDS_ReturnCode_t DDS_StringDataWriter_write_w_timestamp** (**DDS_StringDataWriter** *self, const char *instance_data, const **DDS_InstanceHandle_t** *handle, const struct **DDS_Time_t** *source_timestamp)
*Performs the same function as **DDS_StringDataWriter_write** (p. 1119) except that it also provides the value for the source_timestamp.*
- **DDS_ReturnCode_t DDS_StringDataWriter_write_w_params** (**DDS_StringDataWriter** *self, const char *instance_data, struct **DDS_WriteParams_t** *params)
*Performs the same function as **DDS_StringDataWriter_write** (p. 1119) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.*
- **DDS_StringDataReader * DDS_StringDataReader_narrow** (**DDS_DataReader** *reader)
*Narrow the given **DDS_DataReader** (p. 615) pointer to a **DDS_StringDataReader** (p. 1114) pointer.*
- **DDS_DataReader * DDS_StringDataReader_as_datareader** (**DDS_StringDataReader** *reader)
*Widen the given **DDS_StringDataReader** (p. 1114) pointer to a **DDS_DataReader** (p. 615) pointer.*
- **DDS_ReturnCode_t DDS_StringDataReader_read** (**DDS_StringDataReader** *self, struct **DDS_StringSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, **DDS_SampleStateMask** sample_states, **DDS_ViewStateMask** view_states, **DDS_InstanceStateMask** instance_states)
*Access a collection of data samples from the **DDS_DataReader** (p. 615).*
- **DDS_ReturnCode_t DDS_StringDataReader_take** (**DDS_StringDataReader** *self, struct **DDS_StringSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, **DDS_SampleStateMask** sample_mask, **DDS_ViewStateMask** view_mask, **DDS_InstanceStateMask** instance_mask)
*Access a collection of data-samples from the **DDS_DataReader** (p. 615).*
- **DDS_ReturnCode_t DDS_StringDataReader_read_w_condition** (**DDS_StringDataReader** *self, struct **DDS_StringSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, **DDS_↵ReadCondition** *condition)
*Accesses via **DDS_StringDataReader_read** (p. 1121) the samples that match the criteria specified in the **DDS_Read↵Condition** (p. 694).*
- **DDS_ReturnCode_t DDS_StringDataReader_take_w_condition** (**DDS_StringDataReader** *self, struct **DDS_StringSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, **DDS_↵ReadCondition** *condition)
*Analogous to **DDS_StringDataReader_read_w_condition** (p. 1122) except it accesses samples via the **DDS_String↵DataReader_take** (p. 1121) operation.*
- **DDS_ReturnCode_t DDS_StringDataReader_read_next_sample** (**DDS_StringDataReader** *self, char *received_data, struct **DDS_SampleInfo** *sample_info)
*Copies the next not-previously-accessed data value from the **DDS_DataReader** (p. 615).*
- **DDS_ReturnCode_t DDS_StringDataReader_take_next_sample** (**DDS_StringDataReader** *self, char *received_data, struct **DDS_SampleInfo** *sample_info)
*Copies the next not-previously-accessed data value from the **DDS_DataReader** (p. 615).*
- **DDS_ReturnCode_t DDS_StringDataReader_return_loan** (**DDS_StringDataReader** *self, struct **DDS_↵StringSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq)
*Indicates to the **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615).*

5.123.1 Detailed Description

Built-in type consisting of a single character string.

5.123.2 Typedef Documentation

5.123.2.1 DDS_StringDataWriter

```
typedef struct DDS_StringDataWriter DDS_StringDataWriter
```

<<*interface*>> (p. 1013) Instantiates `DataWriter < char* >`.

See also

FooDataWriter (p. 2113)

DDS_DataWriter (p. 485)

String Support (p. 1526)

5.123.2.2 DDS_StringDataReader

```
typedef struct DDS_StringDataReader DDS_StringDataReader
```

<<*interface*>> (p. 1013) Instantiates `DataReader < char* >`.

See also

FooDataReader (p. 2112)

DDS_DataReader (p. 615)

String Support (p. 1526)

5.123.3 Function Documentation

5.123.3.1 DDS_StringTypeSupport_register_type()

```
DDS_ReturnCode_t DDS_StringTypeSupport_register_type (
    DDS_DomainParticipant * participant,
    const char * type_name )
```

Allows an application to communicate to RTI Connexx 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 **DDS_StringTypeSupport_get_type_name** (p. 1116). 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 function can also be used to register the same **DDS_StringTypeSupport** (p. 2006) with a **DDS_Domain↵Participant** (p. 77) using different values for the `type_name`.

If `register_type` is called multiple times with the same **DDS_DomainParticipant** (p. 77) and `type_name`, the second (and subsequent) registrations are ignored by the operation.

Parameters

<i>participant</i>	<< <i>in</i> >> (p. 1013) the DDS_DomainParticipant (p. 77) to register the data type char* with. Cannot be NULL.
<i>type_name</i>	<< <i>in</i> >> (p. 1013) the type name under with the data type char* is registered with the participant; this type name is used when creating a new DDS_Topic (p. 175). (See DDS_DomainParticipant_create_topic (p. 115).) The name may not be NULL or longer than 255 characters.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

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

DDS_DomainParticipant_create_topic (p. 115)

5.123.3.2 DDS_StringTypeSupport_unregister_type()

```
DDS_ReturnCode_t DDS_StringTypeSupport_unregister_type (
    DDS_DomainParticipant * participant,
    const char * type_name )
```

Allows an application to unregister the char* data type from RTI Connex. 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 **DDS_Topic** (p. 175) objects referencing the type have been destroyed. If the type is not registered with the participant, or if any **DDS_Topic** (p. 175) is associated with the type, the operation will fail with **DDS_RETCODE_ERROR** (p. 1222).

Postcondition

All information about the type is removed from RTI Connex. No further communication using this type is possible.

Parameters

<i>participant</i>	<< <i>in</i> >> (p. 1013) the DDS_DomainParticipant (p. 77) to unregister the data type char* from. Cannot be NULL.
<i>type_name</i>	<< <i>in</i> >> (p. 1013) 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.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_BAD_PARAMETER** (p. 1222) or **DDS_RETCODE_ERROR** (p. 1222)

MT Safety:

SAFE.

See also

DDS_StringTypeSupport_register_type (p. 1114)

5.123.3.3 DDS_StringTypeSupport_get_type_name()

```
const char * DDS_StringTypeSupport_get_type_name (
    void )
```

Get the default name for the char* type.

Can be used for calling **DDS_StringTypeSupport_register_type** (p. 1114) or creating **DDS_Topic** (p. 175).

Returns

default name for the char* type.

See also

DDS_StringTypeSupport_register_type (p. 1114)

DDS_DomainParticipant_create_topic (p. 115)

5.123.3.4 DDS_StringTypeSupport_print_data()

```
void DDS_StringTypeSupport_print_data (
    const char * a_data )
```

<<**extension**>> (p. 1012) Print value of data type to standard out.

The *generated* implementation of the operation knows how to print value of a data type.

Parameters

<i>a_data</i>	<< <i>in</i> >> (p. 1013) String to be printed.

5.123.3.5 DDS_StringTypeSupport_get_typecode()

```
DDS_TypeCode * DDS_StringTypeSupport_get_typecode (
    void )
```

<<*extension*>> (p. 1012) Retrieves the TypeCode for the Type.

See also

FooTypeSupport_get_typecode (p. 226)

5.123.3.6 DDS_StringTypeSupport_serialize_data_to_cdr_buffer()

```
DDS_ReturnCode_t DDS_StringTypeSupport_serialize_data_to_cdr_buffer (
    char * buffer,
    unsigned int * length,
    const char * a_data )
```

<<*extension*>> (p. 1012) Serializes the input sample into a CDR buffer of octets.

See also

FooTypeSupport_serialize_data_to_cdr_buffer (p. 223)

5.123.3.7 DDS_StringTypeSupport_serialize_data_to_cdr_buffer_ex()

```
DDS_ReturnCode_t DDS_StringTypeSupport_serialize_data_to_cdr_buffer_ex (
    char * buffer,
    unsigned int * length,
    const char * a_data,
    DDS_DataRepresentationId_t representation )
```

<<*extension*>> (p. 1012) Serializes the input sample into a buffer of octets.

See also

FooTypeSupport_serialize_data_to_cdr_buffer (p. 223)

5.123.3.8 DDS_StringTypeSupport_deserialize_data_from_cdr_buffer()

```
DDS_ReturnCode_t DDS_StringTypeSupport_deserialize_data_from_cdr_buffer (
    char ** a_data,
    const char * buffer,
    unsigned int length )
```

<<*extension*>> (p. 1012) Deserializes a sample from a buffer of octets.

See also

FooTypeSupport_deserialize_data_from_cdr_buffer (p. 225)

5.123.3.9 DDS_StringTypeSupport_data_to_string()

```
DDS_ReturnCode_t DDS_StringTypeSupport_data_to_string (
    const char * sample,
    char * str,
    DDS_UnsignedLong * str_size,
    DDS_PrintFormatProperty * property )
```

<<*extension*>> (p. 1012) Get the string representation of an input sample.

See also

FooTypeSupport_data_to_string (p. 225)

5.123.3.10 DDS_StringDataWriter_narrow()

```
DDS_StringDataWriter * DDS_StringDataWriter_narrow (
    DDS_DataWriter * writer )
```

Narrow the given **DDS_DataWriter** (p. 485) pointer to a **DDS_StringDataWriter** (p. 1114) pointer.

See also

FooDataWriter_narrow (p. 490)

5.123.3.11 DDS_StringDataWriter_as_datawriter()

```
DDS_DataWriter * DDS_StringDataWriter_as_datawriter (
    DDS_StringDataWriter * writer )
```

Widen the given **DDS_StringDataWriter** (p. 1114) pointer to a **DDS_DataWriter** (p. 485) pointer.

See also

FooDataWriter_as_datawriter (p. 490)

5.123.3.12 DDS_StringDataWriter_create_data()

```
char * DDS_StringDataWriter_create_data (
    DDS_StringDataWriter * self )
```

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 **DDS_StringDataWriter_delete_data** (p. 1119).

Returns

Newly created string data, or NULL on failure.

See also

BuiltinTypeMemoryManagement section of Built-in Types

5.123.3.13 DDS_StringDataWriter_delete_data()

```
DDS_Boolean DDS_StringDataWriter_delete_data (
    DDS_StringDataWriter * self,
    char * sample )
```

Destroys a string data instance created by **DDS_StringDataWriter_create_data** (p. 1119).

Returns

DDS_BOOLEAN_TRUE (p. 1201) upon successful deletion.

5.123.3.14 DDS_StringDataWriter_write()

```
DDS_ReturnCode_t DDS_StringDataWriter_write (
    DDS_StringDataWriter * self,
    const char * instance_data,
    const DDS_InstanceHandle_t * handle )
```

Modifies the value of a string data instance.

See also

FooDataWriter_write (p. 496)

5.123.3.15 DDS_StringDataWriter_write_w_timestamp()

```
DDS_ReturnCode_t DDS_StringDataWriter_write_w_timestamp (
    DDS_StringDataWriter * self,
    const char * instance_data,
    const DDS_InstanceHandle_t * handle,
    const struct DDS_Time_t * source_timestamp )
```

Performs the same function as **DDS_StringDataWriter_write** (p. 1119) except that it also provides the value for the `source_timestamp`.

See also

FooDataWriter_write_w_timestamp (p. 500)

5.123.3.16 DDS_StringDataWriter_write_w_params()

```
DDS_ReturnCode_t DDS_StringDataWriter_write_w_params (
    DDS_StringDataWriter * self,
    const char * instance_data,
    struct DDS_WriteParams_t * params )
```

Performs the same function as **DDS_StringDataWriter_write** (p. 1119) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

See also

FooDataWriter_write_w_params (p. 501)

5.123.3.17 DDS_StringDataReader_narrow()

```
DDS_StringDataReader * DDS_StringDataReader_narrow (
    DDS_DataReader * reader )
```

Narrow the given **DDS_DataReader** (p. 615) pointer to a **DDS_StringDataReader** (p. 1114) pointer.

See also

FooDataReader_narrow (p. 624)

5.123.3.18 DDS_StringDataReader_as_datareader()

```
DDS_DataReader * DDS_StringDataReader_as_datareader (
    DDS_StringDataReader * reader )
```

Widen the given **DDS_StringDataReader** (p. 1114) pointer to a **DDS_DataReader** (p. 615) pointer.

See also

FooDataReader_as_datareader (p. 624)

5.123.3.19 DDS_StringDataReader_read()

```
DDS_ReturnCode_t DDS_StringDataReader_read (
    DDS_StringDataReader * self,
    struct DDS_StringSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    DDS_SampleStateMask sample_states,
    DDS_ViewStateMask view_states,
    DDS_InstanceStateMask instance_states )
```

Access a collection of data samples from the **DDS_DataReader** (p. 615).

See also

FooDataReader_read (p. 625)

5.123.3.20 DDS_StringDataReader_take()

```
DDS_ReturnCode_t DDS_StringDataReader_take (
    DDS_StringDataReader * self,
    struct DDS_StringSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    DDS_SampleStateMask sample_mask,
    DDS_ViewStateMask view_mask,
    DDS_InstanceStateMask instance_mask )
```

Access a collection of data-samples from the **DDS_DataReader** (p. 615).

See also

FooDataReader_take (p. 626)

5.123.3.21 DDS_StringDataReader_read_w_condition()

```
DDS_ReturnCode_t DDS_StringDataReader_read_w_condition (
    DDS_StringDataReader * self,
    struct DDS_StringSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    DDS_ReadCondition * condition )
```

Accesses via **DDS_StringDataReader_read** (p. 1121) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).

See also

FooDataReader_read_w_condition (p. 630)

5.123.3.22 DDS_StringDataReader_take_w_condition()

```
DDS_ReturnCode_t DDS_StringDataReader_take_w_condition (
    DDS_StringDataReader * self,
    struct DDS_StringSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    DDS_ReadCondition * condition )
```

Analogous to **DDS_StringDataReader_read_w_condition** (p. 1122) except it accesses samples via the **DDS_StringDataReader_take** (p. 1121) operation.

See also

FooDataReader_take_w_condition (p. 632)

5.123.3.23 DDS_StringDataReader_read_next_sample()

```
DDS_ReturnCode_t DDS_StringDataReader_read_next_sample (
    DDS_StringDataReader * self,
    char * received_data,
    struct DDS_SampleInfo * sample_info )
```

Copies the next not-previously-accessed data value from the **DDS_DataReader** (p. 615).

See also

FooDataReader_read_next_sample (p. 633)

5.123.3.24 DDS_StringDataReader_take_next_sample()

```
DDS_ReturnCode_t DDS_StringDataReader_take_next_sample (
    DDS_StringDataReader * self,
    char * received_data,
    struct DDS_SampleInfo * sample_info )
```

Copies the next not-previously-accessed data value from the **DDS_DataReader** (p. 615).

See also

FooDataReader_take_next_sample (p. 634)

5.123.3.25 DDS_StringDataReader_return_loan()

```
DDS_ReturnCode_t DDS_StringDataReader_return_loan (
    DDS_StringDataReader * self,
    struct DDS_StringSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq )
```

Indicates to the **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615).

See also

FooDataReader_return_loan (p. 646)

5.124 KeyedString Built-in Type

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

Data Structures

- struct **DDS_KeyedString**
Keyed string built-in type.
- struct **DDS_KeyedStringSeq**
Instantiates `FooSeq` (p. 2113) < `DDS_KeyedString` (p. 1812) > .
- struct **DDS_KeyedStringTypeSupport**
<<interface>> (p. 1013) Keyed string type support.

Typedefs

- typedef struct **DDS_KeyedString** **DDS_KeyedString**
Keyed string built-in type.
- typedef struct **DDS_KeyedStringDataWriter** **DDS_KeyedStringDataWriter**
<<interface>> (p. 1013) Instantiates `DataWriter` < `DDS_KeyedString` (p. 1812) > .
- typedef struct **DDS_KeyedStringDataReader** **DDS_KeyedStringDataReader**
<<interface>> (p. 1013) Instantiates `DataReader` < `DDS_KeyedString` (p. 1812) > .

Functions

- **DDS_KeyedString * DDS_KeyedString_new** (void)
Constructor.
- **DDS_KeyedString * DDS_KeyedString_new_w_size** (int key_size, int size)
Constructor that specifies the allocated sizes.
- void **DDS_KeyedString_delete** (**DDS_KeyedString** *self)
Destructor.
- **DDS_ReturnCode_t DDS_KeyedStringTypeSupport_register_type** (**DDS_DomainParticipant** *participant, const char *type_name)
Allows an application to communicate to RTI Connext the existence of the `DDS_KeyedString` (p. 1812) data type.
- **DDS_ReturnCode_t DDS_KeyedStringTypeSupport_unregister_type** (**DDS_DomainParticipant** *participant, const char *type_name)
Allows an application to unregister the `DDS_KeyedString` (p. 1812) data type from RTI Connext. After calling unregister_type, no further communication using this type is possible.
- const char * **DDS_KeyedStringTypeSupport_get_type_name** (void)
Get the default name for the `DDS_KeyedString` (p. 1812) type.
- void **DDS_KeyedStringTypeSupport_print_data** (const **DDS_KeyedString** *a_data)
<<extension>> (p. 1012) Print value of data type to standard out.
- **DDS_TypeCode * DDS_KeyedStringTypeSupport_get_typecode** (void)
<<extension>> (p. 1012) Retrieves the TypeCode for the Type.
- **DDS_ReturnCode_t DDS_KeyedStringTypeSupport_serialize_data_to_cdr_buffer** (char *buffer, unsigned int *length, const **DDS_KeyedString** *a_data)
<<extension>> (p. 1012) Serializes the input sample into a CDR buffer of octets.
- **DDS_ReturnCode_t DDS_KeyedStringTypeSupport_serialize_data_to_cdr_buffer_ex** (char *buffer, unsigned int *length, const **DDS_KeyedString** *a_data, **DDS_DataRepresentationId_t** representation)
<<extension>> (p. 1012) Serializes the input sample into a buffer of octets.
- **DDS_ReturnCode_t DDS_KeyedStringTypeSupport_deserialize_data_from_cdr_buffer** (**DDS_KeyedString** *a_data, const char *buffer, unsigned int length)

- <<extension>> (p. 1012) Deserializes a sample from a buffer of octets.
- **DDS_ReturnCode_t DDS_KeyedStringTypeSupport_data_to_string** (const **DDS_KeyedString** *sample, char *str, **DDS_UnsignedLong** *str_size, **DDS_PrintFormatProperty** *property)
 <<extension>> (p. 1012) Get the string representation of an input sample.
 - **DDS_KeyedStringDataWriter * DDS_KeyedStringDataWriter_narrow** (**DDS_DataWriter** *writer)
 Narrow the given **DDS_DataWriter** (p. 485) pointer to a **DDS_KeyedStringDataWriter** (p. 1129) pointer.
 - **DDS_DataWriter * DDS_KeyedStringDataWriter_as_datawriter** (**DDS_KeyedStringDataWriter** *writer)
 Widen the given **DDS_KeyedStringDataWriter** (p. 1129) pointer to a **DDS_DataWriter** (p. 485) pointer.
 - **DDS_InstanceHandle_t DDS_KeyedStringDataWriter_register_instance** (**DDS_KeyedStringDataWriter** *self, const **DDS_KeyedString** *instance_data)
 Informs RTI Connex that the application will be modifying a particular instance.
 - **DDS_InstanceHandle_t DDS_KeyedStringDataWriter_register_instance_w_key** (**DDS_KeyedStringDataWriter** *self, const char *key)
 <<extension>> (p. 1012) Informs RTI Connex that the application will be modifying a particular instance.
 - **DDS_InstanceHandle_t DDS_KeyedStringDataWriter_register_instance_w_timestamp** (**DDS_KeyedStringDataWriter** *self, const **DDS_KeyedString** *instance_data, const struct **DDS_Time_t** *source_timestamp)
 Performs the same functions as **DDS_KeyedStringDataWriter_register_instance** (p. 1135) except that the application provides the value for the *source_timestamp*.
 - **DDS_InstanceHandle_t DDS_KeyedStringDataWriter_register_instance_w_key_w_timestamp** (**DDS_KeyedStringDataWriter** *self, const char *key, const struct **DDS_Time_t** *source_timestamp)
 <<extension>> (p. 1012) Performs the same functions as **DDS_KeyedStringDataWriter_register_instance_w_key** (p. 1136) except that the application provides the value for the *source_timestamp*.
 - **DDS_ReturnCode_t DDS_KeyedStringDataWriter_unregister_instance** (**DDS_KeyedStringDataWriter** *self, const **DDS_KeyedString** *instance_data, const **DDS_InstanceHandle_t** *handle)
 Reverses the action of **DDS_KeyedStringDataWriter_register_instance** (p. 1135).
 - **DDS_ReturnCode_t DDS_KeyedStringDataWriter_unregister_instance_w_key** (**DDS_KeyedStringDataWriter** *self, const char *key, const **DDS_InstanceHandle_t** *handle)
 <<extension>> (p. 1012) Reverses the action of **DDS_KeyedStringDataWriter_register_instance_w_key** (p. 1136).
 - **DDS_ReturnCode_t DDS_KeyedStringDataWriter_unregister_instance_w_timestamp** (**DDS_KeyedStringDataWriter** *self, const **DDS_KeyedString** *instance_data, const **DDS_InstanceHandle_t** *handle, const struct **DDS_Time_t** *source_timestamp)
 Performs the same function as **DDS_KeyedStringDataWriter_unregister_instance** (p. 1137) except that it also provides the value for the *source_timestamp*.
 - **DDS_ReturnCode_t DDS_KeyedStringDataWriter_unregister_instance_w_key_w_timestamp** (**DDS_KeyedStringDataWriter** *self, const char *key, const **DDS_InstanceHandle_t** *handle, const struct **DDS_Time_t** *source_timestamp)
 <<extension>> (p. 1012) Performs the same function as **DDS_KeyedStringDataWriter_unregister_instance_w_key** (p. 1137) except that it also provides the value for the *source_timestamp*.
 - **DDS_KeyedString * DDS_KeyedStringDataWriter_create_data** (**DDS_KeyedStringDataWriter** *self)
 Creates a keyed string data instance.
 - **DDS_Boolean DDS_KeyedStringDataWriter_delete_data** (**DDS_KeyedStringDataWriter** *self, **DDS_KeyedString** *sample)
 Destroys a keyed string data instance created by **DDS_KeyedStringDataWriter_create_data** (p. 1138).
 - **DDS_ReturnCode_t DDS_KeyedStringDataWriter_write** (**DDS_KeyedStringDataWriter** *self, const **DDS_KeyedString** *instance_data, const **DDS_InstanceHandle_t** *handle)
 Modifies the value of a **DDS_KeyedString** (p. 1812) data instance.
 - **DDS_ReturnCode_t DDS_KeyedStringDataWriter_write_string_w_key** (**DDS_KeyedStringDataWriter** *self, const char *key, const char *str, const **DDS_InstanceHandle_t** *handle)
 <<extension>> (p. 1012) Modifies the value of a **DDS_KeyedString** (p. 1812) data instance.

- **DDS_ReturnCode_t DDS_KeyedStringDataWriter_write_w_timestamp** (**DDS_KeyedStringDataWriter** *self, const **DDS_KeyedString** *instance_data, const **DDS_InstanceHandle_t** *handle, const struct **DDS_↵**
Time_t *source_timestamp)
*Performs the same function as **DDS_KeyedStringDataWriter_write** (p. 1139) except that it also provides the value for the **source_timestamp**.*
- **DDS_ReturnCode_t DDS_KeyedStringDataWriter_write_string_w_key_w_timestamp** (**DDS_Keyed↵**
StringDataWriter *self, const char *key, const char *str, const **DDS_InstanceHandle_t** *handle, const struct **DDS_Time_t** *source_timestamp)
*<<extension>> (p. 1012) Performs the same function as **DDS_KeyedStringDataWriter_write_string_w_key** (p. 1139) except that it also provides the value for the **source_timestamp**.*
- **DDS_ReturnCode_t DDS_KeyedStringDataWriter_write_w_params** (**DDS_KeyedStringDataWriter** *self, const **DDS_KeyedString** *instance_data, struct **DDS_WriteParams_t** *params)
*Performs the same function as **DDS_KeyedStringDataWriter_write** (p. 1139) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.*
- **DDS_ReturnCode_t DDS_KeyedStringDataWriter_write_string_w_key_w_params** (**DDS_KeyedString↵**
DataWriter *self, const char *key, const char *str, struct **DDS_WriteParams_t** *params)
*<<extension>> (p. 1012) Performs the same function as **DDS_KeyedStringDataWriter_write_string_w_key** (p. 1139) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.*
- **DDS_ReturnCode_t DDS_KeyedStringDataWriter_dispose** (**DDS_KeyedStringDataWriter** *self, const **DDS_KeyedString** *instance_data, const **DDS_InstanceHandle_t** *instance_handle)
Requests the middleware to delete the data.
- **DDS_ReturnCode_t DDS_KeyedStringDataWriter_dispose_w_key** (**DDS_KeyedStringDataWriter** *self, const char *key, const **DDS_InstanceHandle_t** *instance_handle)
<<extension>> (p. 1012) Requests the middleware to delete the data.
- **DDS_ReturnCode_t DDS_KeyedStringDataWriter_dispose_w_timestamp** (**DDS_KeyedStringDataWriter** *self, const **DDS_KeyedString** *instance_data, const **DDS_InstanceHandle_t** *instance_handle, const struct **DDS_Time_t** *source_timestamp)
*Performs the same functions as **DDS_KeyedStringDataWriter_dispose** (p. 1141) except that the application provides the value for the **source_timestamp** that is made available to **DDS_DataReader** (p. 615) objects by means of the **source_timestamp** attribute inside the **DDS_SampleInfo** (p. 1985).*
- **DDS_ReturnCode_t DDS_KeyedStringDataWriter_dispose_w_key_w_timestamp** (**DDS_KeyedString↵**
DataWriter *self, const char *key, const **DDS_InstanceHandle_t** *instance_handle, const struct **DDS_Time_t** *source_timestamp)
*<<extension>> (p. 1012) Performs the same functions as **DDS_KeyedStringDataWriter_dispose_w_key** (p. 1141) except that the application provides the value for the **source_timestamp** that is made available to **DDS_DataReader** (p. 615) objects by means of the **source_timestamp** attribute inside the **DDS_SampleInfo** (p. 1985).*
- **DDS_ReturnCode_t DDS_KeyedStringDataWriter_get_key_value** (**DDS_KeyedStringDataWriter** *self, **DDS_KeyedString** *key_holder, const **DDS_InstanceHandle_t** *handle)
*Retrieve the instance **key** that corresponds to an instance **handle**.*
- **DDS_ReturnCode_t DDS_KeyedStringDataWriter_get_key_value_w_key** (**DDS_KeyedStringDataWriter** *self, char *key, const **DDS_InstanceHandle_t** *handle)
*<<extension>> (p. 1012) Retrieve the instance **key** that corresponds to an instance **handle**.*
- **DDS_InstanceHandle_t DDS_KeyedStringDataWriter_lookup_instance** (**DDS_KeyedStringDataWriter** *self, const **DDS_KeyedString** *key_holder)
*Retrieve the instance **handle** that corresponds to an instance **key_holder**.*
- **DDS_InstanceHandle_t DDS_KeyedStringDataWriter_lookup_instance_w_key** (**DDS_KeyedStringData↵**
Writer *self, const char *key)
*<<extension>> (p. 1012) Retrieve the instance **handle** that corresponds to an instance **key**.*
- **DDS_KeyedStringDataReader * DDS_KeyedStringDataReader_narrow** (**DDS_DataReader** *reader)
*Narrow the given **DDS_DataReader** (p. 615) pointer to a **DDS_KeyedStringDataReader** (p. 1129) pointer.*

- **DDS_DataReader * DDS_KeyedStringDataReader_as_datareader (DDS_KeyedStringDataReader *reader)**

*Widen the given **DDS_KeyedStringDataReader** (p. 1129) pointer to a **DDS_DataReader** (p. 615) pointer.*

- **DDS_ReturnCode_t DDS_KeyedStringDataReader_read (DDS_KeyedStringDataReader *self, struct DDS_KeyedStringSeq *received_data, struct DDS_SampleInfoSeq *info_seq, DDS_Long max_samples, DDS_SampleStateMask sample_states, DDS_ViewStateMask view_states, DDS_InstanceStateMask instance_states)**

*Access a collection of data samples from the **DDS_DataReader** (p. 615).*

- **DDS_ReturnCode_t DDS_KeyedStringDataReader_take (DDS_KeyedStringDataReader *self, struct DDS_KeyedStringSeq *received_data, struct DDS_SampleInfoSeq *info_seq, DDS_Long max_samples, DDS_SampleStateMask sample_mask, DDS_ViewStateMask view_mask, DDS_InstanceStateMask instance_mask)**

*Access a collection of data-samples from the **DDS_DataReader** (p. 615).*

- **DDS_ReturnCode_t DDS_KeyedStringDataReader_read_w_condition (DDS_KeyedStringDataReader *self, struct DDS_KeyedStringSeq *received_data, struct DDS_SampleInfoSeq *info_seq, DDS_Long max←_samples, DDS_ReadCondition *condition)**

*Accesses via **DDS_KeyedStringDataReader_read** (p. 1144) the samples that match the criteria specified in the **DDS←_ReadCondition** (p. 694).*

- **DDS_ReturnCode_t DDS_KeyedStringDataReader_take_w_condition (DDS_KeyedStringDataReader *self, struct DDS_KeyedStringSeq *received_data, struct DDS_SampleInfoSeq *info_seq, DDS_Long max←_samples, DDS_ReadCondition *condition)**

*Analogous to **DDS_KeyedStringDataReader_read_w_condition** (p. 1145) except it accesses samples via the **DDS←_KeyedStringDataReader_take** (p. 1144) operation.*

- **DDS_ReturnCode_t DDS_KeyedStringDataReader_read_next_sample (DDS_KeyedStringDataReader *self, DDS_KeyedString *received_data, struct DDS_SampleInfo *sample_info)**

*Copies the next not-previously-accessed data value from the **DDS_DataReader** (p. 615).*

- **DDS_ReturnCode_t DDS_KeyedStringDataReader_take_next_sample (DDS_KeyedStringDataReader *self, DDS_KeyedString *received_data, struct DDS_SampleInfo *sample_info)**

*Copies the next not-previously-accessed data value from the **DDS_DataReader** (p. 615).*

- **DDS_ReturnCode_t DDS_KeyedStringDataReader_read_instance (DDS_KeyedStringDataReader *self, struct DDS_KeyedStringSeq *received_data, struct DDS_SampleInfoSeq *info_seq, DDS_Long max←_samples, const DDS_InstanceHandle_t *a_handle, DDS_SampleStateMask sample_states, DDS_View←_StateMask view_states, DDS_InstanceStateMask instance_states)**

*Access a collection of data samples from the **DDS_DataReader** (p. 615).*

- **DDS_ReturnCode_t DDS_KeyedStringDataReader_take_instance (DDS_KeyedStringDataReader *self, struct DDS_KeyedStringSeq *received_data, struct DDS_SampleInfoSeq *info_seq, DDS_Long max←_samples, const DDS_InstanceHandle_t *a_handle, DDS_SampleStateMask sample_states, DDS_View←_StateMask view_states, DDS_InstanceStateMask instance_states)**

*Access a collection of data samples from the **DDS_DataReader** (p. 615).*

- **DDS_ReturnCode_t DDS_KeyedStringDataReader_read_instance_w_condition (DDS_KeyedString←_DataReader *self, struct DDS_KeyedStringSeq *received_data, struct DDS_SampleInfoSeq *info_seq, DDS_Long max←_samples, const DDS_InstanceHandle_t *a_handle, DDS_ReadCondition *condition)**

*Accesses via **DDS_KeyedStringDataReader_read_instance** (p. 1146) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).*

- **DDS_ReturnCode_t DDS_KeyedStringDataReader_take_instance_w_condition (DDS_KeyedString←_DataReader *self, struct DDS_KeyedStringSeq *received_data, struct DDS_SampleInfoSeq *info_seq, DDS_Long max←_samples, const DDS_InstanceHandle_t *a_handle, DDS_ReadCondition *condition)**

*Accesses via **DDS_KeyedStringDataReader_take_instance** (p. 1146) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).*

- **DDS_ReturnCode_t DDS_KeyedStringDataReader_read_next_instance** (**DDS_KeyedStringDataReader** *self, struct **DDS_KeyedStringSeq** *received_data, struct **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)
*Access a collection of data samples from the **DDS_DataReader** (p. 615).*
- **DDS_ReturnCode_t DDS_KeyedStringDataReader_take_next_instance** (**DDS_KeyedStringDataReader** *self, struct **DDS_KeyedStringSeq** *received_data, struct **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)
*Access a collection of data samples from the **DDS_DataReader** (p. 615).*
- **DDS_ReturnCode_t DDS_KeyedStringDataReader_read_next_instance_w_condition** (**DDS_Keyed←StringDataReader** *self, struct **DDS_KeyedStringSeq** *received_data, struct **DDS_SampleInfoSeq** *info←_seq, **DDS_Long** max_samples, const **DDS_InstanceHandle_t** *previous_handle, **DDS_ReadCondition** *condition)
*Accesses via **DDS_KeyedStringDataReader_read_next_instance** (p. 1147) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).*
- **DDS_ReturnCode_t DDS_KeyedStringDataReader_take_next_instance_w_condition** (**DDS_Keyed←StringDataReader** *self, struct **DDS_KeyedStringSeq** *received_data, struct **DDS_SampleInfoSeq** *info←_seq, **DDS_Long** max_samples, const **DDS_InstanceHandle_t** *previous_handle, **DDS_ReadCondition** *condition)
*Accesses via **DDS_KeyedStringDataReader_take_next_instance** (p. 1148) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).*
- **DDS_ReturnCode_t DDS_KeyedStringDataReader_return_loan** (**DDS_KeyedStringDataReader** *self, struct **DDS_KeyedStringSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq)
*Indicates to the **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615).*
- **DDS_ReturnCode_t DDS_KeyedStringDataReader_get_key_value** (**DDS_KeyedStringDataReader** *self, **DDS_KeyedString** *key_holder, const **DDS_InstanceHandle_t** *handle)
*Retrieve the instance *key* that corresponds to an instance *handle*.*
- **DDS_ReturnCode_t DDS_KeyedStringDataReader_get_key_value_w_key** (**DDS_KeyedStringData←Reader** *self, char *key, const **DDS_InstanceHandle_t** *handle)
*<<extension>> (p. 1012) Retrieve the instance *key* that corresponds to an instance *handle*.*
- **DDS_InstanceHandle_t DDS_KeyedStringDataReader_lookup_instance** (**DDS_KeyedStringDataReader** *self, const **DDS_KeyedString** *key_holder)
*Retrieve the instance *handle* that corresponds to an instance *key_holder*.*
- **DDS_InstanceHandle_t DDS_KeyedStringDataReader_lookup_instance_w_key** (**DDS_KeyedString←DataReader** *self, const char *key)
*<<extension>> (p. 1012) Retrieve the instance *handle* that corresponds to an instance *key*.*

Variables

- char * **DDS_KeyedString::key**
Instance key associated with the specified value.
- char * **DDS_KeyedString::value**
String value.

5.124.1 Detailed Description

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

5.124.2 Typedef Documentation

5.124.2.1 DDS_KeyedString

```
typedef struct DDS_KeyedString DDS_KeyedString
```

Keyed string built-in type.

5.124.2.2 DDS_KeyedStringDataWriter

```
typedef struct DDS_KeyedStringDataWriter DDS_KeyedStringDataWriter
```

<<**interface**>> (p. 1013) Instantiates `DataWriter` < **DDS_KeyedString** (p. 1812) >.

See also

FooDataWriter (p. 2113)

DDS_DataWriter (p. 485)

5.124.2.3 DDS_KeyedStringDataReader

```
typedef struct DDS_KeyedStringDataReader DDS_KeyedStringDataReader
```

<<**interface**>> (p. 1013) Instantiates `DataReader` < **DDS_KeyedString** (p. 1812) >.

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. 1528); 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. 2112)

DDS_DataReader (p. 615)

5.124.3 Function Documentation

5.124.3.1 DDS_KeyedString_new()

```
DDS_KeyedString * DDS_KeyedString_new (
    void )
```

Constructor.

The default constructor initializes the newly created object with NULL key and value.

Returns

A new **DDS_KeyedString** (p. 1812) or NULL if failure.

5.124.3.2 DDS_KeyedString_new_w_size()

```
DDS_KeyedString * DDS_KeyedString_new_w_size (
    int key_size,
    int size )
```

Constructor that specifies the allocated sizes.

The allocated strings are initialized to empty ("").

Parameters

<i>key_size</i>	<< <i>in</i> >> (p. 1013) Size of the allocated key string (with NULL-terminated character). Cannot be smaller than one.
<i>size</i>	<< <i>in</i> >> (p. 1013) Size of the allocated value string (with NULL-terminated character). Cannot be smaller than one.

Returns

A new **DDS_KeyedString** (p. 1812) or NULL if failure.

5.124.3.3 DDS_KeyedString_delete()

```
void DDS_KeyedString_delete (
    DDS_KeyedString * self )
```

Destructor.

5.124.3.4 DDS_KeyedStringTypeSupport_register_type()

```
DDS_ReturnCode_t DDS_KeyedStringTypeSupport_register_type (
    DDS_DomainParticipant * participant,
    const char * type_name )
```

Allows an application to communicate to RTI Connext the existence of the **DDS_KeyedString** (p. 1812) data type.

By default, The **DDS_KeyedString** (p. 1812) built-in type is automatically registered when a DomainParticipant is created using the type_name returned by **DDS_KeyedStringTypeSupport_get_type_name** (p. 1132). 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 function can also be used to register the same **DDS_KeyedStringTypeSupport** (p. 1813) with a **DDS_DomainParticipant** (p. 77) using different values for the type_name.

If register_type is called multiple times with the same **DDS_DomainParticipant** (p. 77) and type_name, the second (and subsequent) registrations are ignored by the operation.

Parameters

<i>participant</i>	<<in>> (p. 1013) the DDS_DomainParticipant (p. 77) to register the data type DDS_KeyedString (p. 1812) with. Cannot be NULL.
<i>type_name</i>	<<in>> (p. 1013) the type name under with the data type DDS_KeyedString (p. 1812) is registered with the participant; this type name is used when creating a new DDS_Topic (p. 175). (See DDS_DomainParticipant_create_topic (p. 115).) The name may not be NULL or longer than 255 characters.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

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

DDS_DomainParticipant_create_topic (p. 115)

5.124.3.5 DDS_KeyedStringTypeSupport_unregister_type()

```
DDS_ReturnCode_t DDS_KeyedStringTypeSupport_unregister_type (
    DDS_DomainParticipant * participant,
    const char * type_name )
```

Allows an application to unregister the **DDS_KeyedString** (p.1812) data type from RTI Connex. After calling `unregister_type`, no further communication using this type is possible.

Precondition

The **DDS_KeyedString** (p.1812) type with `type_name` is registered with the participant and all **DDS_Topic** (p.175) objects referencing the type have been destroyed. If the type is not registered with the participant, or if any **DDS_Topic** (p.175) is associated with the type, the operation will fail with **DDS_RETCODE_ERROR** (p.1222).

Postcondition

All information about the type is removed from RTI Connex. No further communication using this type is possible.

Parameters

<i>participant</i>	<< <i>in</i> >> (p.1013) the DDS_DomainParticipant (p.77) to unregister the data type DDS_KeyedString (p.1812) from. Cannot be NULL.
<i>type_name</i>	<< <i>in</i> >> (p.1013) the type name under with the data type DDS_KeyedString (p.1812) 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.

Returns

One of the **Standard Return Codes** (p.1221), **DDS_RETCODE_BAD_PARAMETER** (p.1222) or **DDS_RETCODE_ERROR** (p.1222)

MT Safety:

SAFE.

See also

DDS_KeyedStringTypeSupport_register_type (p.1130)

5.124.3.6 DDS_KeyedStringTypeSupport_get_type_name()

```
const char * DDS_KeyedStringTypeSupport_get_type_name (
    void )
```

Get the default name for the **DDS_KeyedString** (p. 1812) type.

Can be used for calling **DDS_KeyedStringTypeSupport_register_type** (p. 1130) or creating **DDS_Topic** (p. 175).

Returns

default name for the **DDS_KeyedString** (p. 1812) type.

See also

DDS_KeyedStringTypeSupport_register_type (p. 1130)

DDS_DomainParticipant_create_topic (p. 115)

5.124.3.7 DDS_KeyedStringTypeSupport_print_data()

```
void DDS_KeyedStringTypeSupport_print_data (
    const DDS_KeyedString * a_data )
```

<<**extension**>> (p. 1012) Print value of data type to standard out.

The *generated* implementation of the operation knows how to print value of a data type.

Parameters

<i>a_data</i>	<< in >> (p. 1013) DDS_KeyedString (p. 1812) to be printed.

5.124.3.8 DDS_KeyedStringTypeSupport_get_typecode()

```
DDS_TypeCode * DDS_KeyedStringTypeSupport_get_typecode (
    void )
```

<<**extension**>> (p. 1012) Retrieves the TypeCode for the Type.

See also

FooTypeSupport_get_typecode (p. 226)

5.124.3.9 DDS_KeyedStringTypeSupport_serialize_data_to_cdr_buffer()

```
DDS_ReturnCode_t DDS_KeyedStringTypeSupport_serialize_data_to_cdr_buffer (
    char * buffer,
    unsigned int * length,
    const DDS_KeyedString * a_data )
```

<<**extension**>> (p. 1012) Serializes the input sample into a CDR buffer of octets.

See also

FooTypeSupport_serialize_data_to_cdr_buffer (p. 223)

5.124.3.10 DDS_KeyedStringTypeSupport_serialize_data_to_cdr_buffer_ex()

```
DDS_ReturnCode_t DDS_KeyedStringTypeSupport_serialize_data_to_cdr_buffer_ex (
    char * buffer,
    unsigned int * length,
    const DDS_KeyedString * a_data,
    DDS_DataRepresentationId_t representation )
```

<<**extension**>> (p. 1012) Serializes the input sample into a buffer of octets.

See also

FooTypeSupport_serialize_data_to_cdr_buffer (p. 223)

5.124.3.11 DDS_KeyedStringTypeSupport_deserialize_data_from_cdr_buffer()

```
DDS_ReturnCode_t DDS_KeyedStringTypeSupport_deserialize_data_from_cdr_buffer (
    DDS_KeyedString * a_data,
    const char * buffer,
    unsigned int length )
```

<<**extension**>> (p. 1012) Deserializes a sample from a buffer of octets.

See also

FooTypeSupport_deserialize_data_from_cdr_buffer (p. 225)

5.124.3.12 DDS_KeyedStringTypeSupport_data_to_string()

```
DDS_ReturnCode_t DDS_KeyedStringTypeSupport_data_to_string (
    const DDS_KeyedString * sample,
    char * str,
    DDS_UnsignedLong * str_size,
    DDS_PrintfFormatProperty * property )
```

<<**extension**>> (p. 1012) Get the string representation of an input sample.

See also

FooTypeSupport_data_to_string (p. 225)

5.124.3.13 DDS_KeyedStringDataWriter_narrow()

```
DDS_KeyedStringDataWriter * DDS_KeyedStringDataWriter_narrow (
    DDS_DataWriter * writer )
```

Narrow the given **DDS_DataWriter** (p. 485) pointer to a **DDS_KeyedStringDataWriter** (p. 1129) pointer.

See also

FooDataWriter_narrow (p. 490)

5.124.3.14 DDS_KeyedStringDataWriter_as_datawriter()

```
DDS_DataWriter * DDS_KeyedStringDataWriter_as_datawriter (
    DDS_KeyedStringDataWriter * writer )
```

Widen the given **DDS_KeyedStringDataWriter** (p. 1129) pointer to a **DDS_DataWriter** (p. 485) pointer.

See also

FooDataWriter_as_datawriter (p. 490)

5.124.3.15 DDS_KeyedStringDataWriter_register_instance()

```
DDS_InstanceHandle_t DDS_KeyedStringDataWriter_register_instance (
    DDS_KeyedStringDataWriter * self,
    const DDS_KeyedString * instance_data )
```

Informs RTI Connexx that the application will be modifying a particular instance.

See also

FooDataWriter_register_instance (p. 491)

5.124.3.16 DDS_KeyedStringDataWriter_register_instance_w_key()

```
DDS_InstanceHandle_t DDS_KeyedStringDataWriter_register_instance_w_key (
    DDS_KeyedStringDataWriter * self,
    const char * key )
```

<<**extension**>> (p. 1012) Informes RTI Connexx that the application will be modifying a particular instance.

See also

FooDataWriter_register_instance (p. 491)

5.124.3.17 DDS_KeyedStringDataWriter_register_instance_w_timestamp()

```
DDS_InstanceHandle_t DDS_KeyedStringDataWriter_register_instance_w_timestamp (
    DDS_KeyedStringDataWriter * self,
    const DDS_KeyedString * instance_data,
    const struct DDS_Time_t * source_timestamp )
```

Performs the same functions as **DDS_KeyedStringDataWriter_register_instance** (p. 1135) except that the application provides the value for the `source_timestamp`.

See also

FooDataWriter_register_instance_w_timestamp (p. 492)

5.124.3.18 DDS_KeyedStringDataWriter_register_instance_w_key_w_timestamp()

```
DDS_InstanceHandle_t DDS_KeyedStringDataWriter_register_instance_w_key_w_timestamp (
    DDS_KeyedStringDataWriter * self,
    const char * key,
    const struct DDS_Time_t * source_timestamp )
```

<<*extension*>> (p. 1012) Performs the same functions as **DDS_KeyedStringDataWriter_register_instance_w_key** (p. 1136) except that the application provides the value for the `source_timestamp`.

See also

FooDataWriter_register_instance_w_timestamp (p. 492)

5.124.3.19 DDS_KeyedStringDataWriter_unregister_instance()

```
DDS_ReturnCode_t DDS_KeyedStringDataWriter_unregister_instance (
    DDS_KeyedStringDataWriter * self,
    const DDS_KeyedString * instance_data,
    const DDS_InstanceHandle_t * handle )
```

Reverses the action of **DDS_KeyedStringDataWriter_register_instance** (p. 1135).

See also

FooDataWriter_unregister_instance (p. 493)

5.124.3.20 DDS_KeyedStringDataWriter_unregister_instance_w_key()

```
DDS_ReturnCode_t DDS_KeyedStringDataWriter_unregister_instance_w_key (
    DDS_KeyedStringDataWriter * self,
    const char * key,
    const DDS_InstanceHandle_t * handle )
```

<<*extension*>> (p. 1012) Reverses the action of **DDS_KeyedStringDataWriter_register_instance_w_key** (p. 1136).

See also

FooDataWriter_unregister_instance (p. 493)

5.124.3.21 DDS_KeyedStringDataWriter_unregister_instance_w_timestamp()

```
DDS_ReturnCode_t DDS_KeyedStringDataWriter_unregister_instance_w_timestamp (
    DDS_KeyedStringDataWriter * self,
    const DDS_KeyedString * instance_data,
    const DDS_InstanceHandle_t * handle,
    const struct DDS_Time_t * source_timestamp )
```

Performs the same function as **DDS_KeyedStringDataWriter_unregister_instance** (p. 1137) except that it also provides the value for the `source_timestamp`.

See also

FooDataWriter_unregister_instance_w_timestamp (p. 495)

5.124.3.22 DDS_KeyedStringDataWriter_unregister_instance_w_key_w_timestamp()

```
DDS_ReturnCode_t DDS_KeyedStringDataWriter_unregister_instance_w_key_w_timestamp (
    DDS_KeyedStringDataWriter * self,
    const char * key,
    const DDS_InstanceHandle_t * handle,
    const struct DDS_Time_t * source_timestamp )
```

<<*extension*>> (p. 1012) Performs the same function as **DDS_KeyedStringDataWriter_unregister_instance_w_key** (p. 1137) except that it also provides the value for the `source_timestamp`.

See also

FooDataWriter_unregister_instance_w_timestamp (p. 495)

5.124.3.23 DDS_KeyedStringDataWriter_create_data()

```
DDS_KeyedString * DDS_KeyedStringDataWriter_create_data (
    DDS_KeyedStringDataWriter * self )
```

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 **DDS_KeyedStringDataWriter_delete_data** (p. 1138).

Returns

Newly created keyed string data, or NULL on failure.

See also

BuiltinTypeMemoryManagement section of Built-in Types

5.124.3.24 DDS_KeyedStringDataWriter_delete_data()

```
DDS_Boolean DDS_KeyedStringDataWriter_delete_data (
    DDS_KeyedStringDataWriter * self,
    DDS_KeyedString * sample )
```

Destroys a keyed string data instance created by **DDS_KeyedStringDataWriter_create_data** (p. 1138).

Returns

DDS_BOOLEAN_TRUE (p. 1201) upon successful deletion.

5.124.3.25 DDS_KeyedStringDataWriter_write()

```
DDS_ReturnCode_t DDS_KeyedStringDataWriter_write (
    DDS_KeyedStringDataWriter * self,
    const DDS_KeyedString * instance_data,
    const DDS_InstanceHandle_t * handle )
```

Modifies the value of a **DDS_KeyedString** (p. 1812) data instance.

See also

FooDataWriter_write (p. 496)

5.124.3.26 DDS_KeyedStringDataWriter_write_string_w_key()

```
DDS_ReturnCode_t DDS_KeyedStringDataWriter_write_string_w_key (
    DDS_KeyedStringDataWriter * self,
    const char * key,
    const char * str,
    const DDS_InstanceHandle_t * handle )
```

<<*extension*>> (p. 1012) Modifies the value of a **DDS_KeyedString** (p. 1812) data instance.

See also

FooDataWriter_write (p. 496)

5.124.3.27 DDS_KeyedStringDataWriter_write_w_timestamp()

```
DDS_ReturnCode_t DDS_KeyedStringDataWriter_write_w_timestamp (
    DDS_KeyedStringDataWriter * self,
    const DDS_KeyedString * instance_data,
    const DDS_InstanceHandle_t * handle,
    const struct DDS_Time_t * source_timestamp )
```

Performs the same function as **DDS_KeyedStringDataWriter_write** (p. 1139) except that it also provides the value for the `source_timestamp`.

See also

FooDataWriter_write_w_timestamp (p. 500)

5.124.3.28 DDS_KeyedStringDataWriter_write_string_w_key_w_timestamp()

```
DDS_ReturnCode_t DDS_KeyedStringDataWriter_write_string_w_key_w_timestamp (
    DDS_KeyedStringDataWriter * self,
    const char * key,
    const char * str,
    const DDS_InstanceHandle_t * handle,
    const struct DDS_Time_t * source_timestamp )
```

<<*extension*>> (p. 1012) Performs the same function as **DDS_KeyedStringDataWriter_write_string_w_key** (p. 1139) except that it also provides the value for the `source_timestamp`.

See also

FooDataWriter_write_w_timestamp (p. 500)

5.124.3.29 DDS_KeyedStringDataWriter_write_w_params()

```
DDS_ReturnCode_t DDS_KeyedStringDataWriter_write_w_params (
    DDS_KeyedStringDataWriter * self,
    const DDS_KeyedString * instance_data,
    struct DDS_WriteParams_t * params )
```

Performs the same function as **DDS_KeyedStringDataWriter_write** (p. 1139) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

See also

FooDataWriter_write_w_params (p. 501)

5.124.3.30 DDS_KeyedStringDataWriter_write_string_w_key_w_params()

```
DDS_ReturnCode_t DDS_KeyedStringDataWriter_write_string_w_key_w_params (
    DDS_KeyedStringDataWriter * self,
    const char * key,
    const char * str,
    struct DDS_WriteParams_t * params )
```

<<*extension*>> (p. 1012) Performs the same function as **DDS_KeyedStringDataWriter_write_string_w_key** (p. 1139) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

See also

FooDataWriter_write_w_params (p. 501)

5.124.3.31 DDS_KeyedStringDataWriter_dispose()

```
DDS_ReturnCode_t DDS_KeyedStringDataWriter_dispose (
    DDS_KeyedStringDataWriter * self,
    const DDS_KeyedString * instance_data,
    const DDS_InstanceHandle_t * instance_handle )
```

Requests the middleware to delete the data.

See also

FooDataWriter_dispose (p. 502)

5.124.3.32 DDS_KeyedStringDataWriter_dispose_w_key()

```
DDS_ReturnCode_t DDS_KeyedStringDataWriter_dispose_w_key (
    DDS_KeyedStringDataWriter * self,
    const char * key,
    const DDS_InstanceHandle_t * instance_handle )
```

<<*extension*>> (p. 1012) Requests the middleware to delete the data.

See also

FooDataWriter_dispose (p. 502)

5.124.3.33 DDS_KeyedStringDataWriter_dispose_w_timestamp()

```
DDS_ReturnCode_t DDS_KeyedStringDataWriter_dispose_w_timestamp (
    DDS_KeyedStringDataWriter * self,
    const DDS_KeyedString * instance_data,
    const DDS_InstanceHandle_t * instance_handle,
    const struct DDS_Time_t * source_timestamp )
```

Performs the same functions as **DDS_KeyedStringDataWriter_dispose** (p. 1141) except that the application provides the value for the `source_timestamp` that is made available to **DDS_DataReader** (p. 615) objects by means of the `source_timestamp` attribute inside the **DDS_SampleInfo** (p. 1985).

See also

FooDataWriter_dispose_w_timestamp (p. 503)

5.124.3.34 DDS_KeyedStringDataWriter_dispose_w_key_w_timestamp()

```
DDS_ReturnCode_t DDS_KeyedStringDataWriter_dispose_w_key_w_timestamp (
    DDS_KeyedStringDataWriter * self,
    const char * key,
    const DDS_InstanceHandle_t * instance_handle,
    const struct DDS_Time_t * source_timestamp )
```

<<*extension*>> (p. 1012) Performs the same functions as **DDS_KeyedStringDataWriter_dispose_w_key** (p. 1141) except that the application provides the value for the `source_timestamp` that is made available to **DDS_DataReader** (p. 615) objects by means of the `source_timestamp` attribute inside the **DDS_SampleInfo** (p. 1985).

See also

FooDataWriter_dispose_w_timestamp (p. 503)

5.124.3.35 DDS_KeyedStringDataWriter_get_key_value()

```
DDS_ReturnCode_t DDS_KeyedStringDataWriter_get_key_value (
    DDS_KeyedStringDataWriter * self,
    DDS_KeyedString * key_holder,
    const DDS_InstanceHandle_t * handle )
```

Retrieve the instance key that corresponds to an instance handle.

See also

FooDataWriter_get_key_value (p. 505)

5.124.3.36 DDS_KeyedStringDataWriter_get_key_value_w_key()

```
DDS_ReturnCode_t DDS_KeyedStringDataWriter_get_key_value_w_key (
    DDS_KeyedStringDataWriter * self,
    char * key,
    const DDS_InstanceHandle_t * handle )
```

<<*extension*>> (p. 1012) Retrieve the instance `key` that corresponds to an instance `handle`.

See also

FooDataWriter_get_key_value (p. 505)

5.124.3.37 DDS_KeyedStringDataWriter_lookup_instance()

```
DDS_InstanceHandle_t DDS_KeyedStringDataWriter_lookup_instance (
    DDS_KeyedStringDataWriter * self,
    const DDS_KeyedString * key_holder )
```

Retrieve the instance `handle` that corresponds to an instance `key_holder`.

See also

FooDataWriter_lookup_instance (p. 506)

5.124.3.38 DDS_KeyedStringDataWriter_lookup_instance_w_key()

```
DDS_InstanceHandle_t DDS_KeyedStringDataWriter_lookup_instance_w_key (
    DDS_KeyedStringDataWriter * self,
    const char * key )
```

<<*extension*>> (p. 1012) Retrieve the instance `handle` that corresponds to an instance `key`.

See also

FooDataWriter_lookup_instance (p. 506)

5.124.3.39 DDS_KeyedStringDataReader_narrow()

```
DDS_KeyedStringDataReader * DDS_KeyedStringDataReader_narrow (
    DDS_DataReader * reader )
```

Narrow the given **DDS_DataReader** (p. 615) pointer to a **DDS_KeyedStringDataReader** (p. 1129) pointer.

See also

FooDataReader_narrow (p. 624)

5.124.3.40 DDS_KeyedStringDataReader_as_datareader()

```
DDS_DataReader * DDS_KeyedStringDataReader_as_datareader (
    DDS_KeyedStringDataReader * reader )
```

Widen the given **DDS_KeyedStringDataReader** (p. 1129) pointer to a **DDS_DataReader** (p. 615) pointer.

See also

FooDataReader_as_datareader (p. 624)

5.124.3.41 DDS_KeyedStringDataReader_read()

```
DDS_ReturnCode_t DDS_KeyedStringDataReader_read (
    DDS_KeyedStringDataReader * self,
    struct DDS_KeyedStringSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    DDS_SampleStateMask sample_states,
    DDS_ViewStateMask view_states,
    DDS_InstanceStateMask instance_states )
```

Access a collection of data samples from the **DDS_DataReader** (p. 615).

See also

FooDataReader_read (p. 625)

5.124.3.42 DDS_KeyedStringDataReader_take()

```

DDS_ReturnCode_t DDS_KeyedStringDataReader_take (
    DDS_KeyedStringDataReader * self,
    struct DDS_KeyedStringSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    DDS_SampleStateMask sample_mask,
    DDS_ViewStateMask view_mask,
    DDS_InstanceStateMask instance_mask )

```

Access a collection of data-samples from the **DDS_DataReader** (p. 615).

See also

FooDataReader_take (p. 626)

5.124.3.43 DDS_KeyedStringDataReader_read_w_condition()

```

DDS_ReturnCode_t DDS_KeyedStringDataReader_read_w_condition (
    DDS_KeyedStringDataReader * self,
    struct DDS_KeyedStringSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    DDS_ReadCondition * condition )

```

Accesses via **DDS_KeyedStringDataReader_read** (p. 1144) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).

See also

FooDataReader_read_w_condition (p. 630)

5.124.3.44 DDS_KeyedStringDataReader_take_w_condition()

```

DDS_ReturnCode_t DDS_KeyedStringDataReader_take_w_condition (
    DDS_KeyedStringDataReader * self,
    struct DDS_KeyedStringSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    DDS_ReadCondition * condition )

```

Analogous to **DDS_KeyedStringDataReader_read_w_condition** (p. 1145) except it accesses samples via the **DDS_KeyedStringDataReader_take** (p. 1144) operation.

See also

FooDataReader_take_w_condition (p. 632)

5.124.3.45 DDS_KeyedStringDataReader_read_next_sample()

```
DDS_ReturnCode_t DDS_KeyedStringDataReader_read_next_sample (
    DDS_KeyedStringDataReader * self,
    DDS_KeyedString * received_data,
    struct DDS_SampleInfo * sample_info )
```

Copies the next not-previously-accessed data value from the **DDS_DataReader** (p. 615).

See also

FooDataReader_read_next_sample (p. 633)

5.124.3.46 DDS_KeyedStringDataReader_take_next_sample()

```
DDS_ReturnCode_t DDS_KeyedStringDataReader_take_next_sample (
    DDS_KeyedStringDataReader * self,
    DDS_KeyedString * received_data,
    struct DDS_SampleInfo * sample_info )
```

Copies the next not-previously-accessed data value from the **DDS_DataReader** (p. 615).

See also

FooDataReader_take_next_sample (p. 634)

5.124.3.47 DDS_KeyedStringDataReader_read_instance()

```
DDS_ReturnCode_t DDS_KeyedStringDataReader_read_instance (
    DDS_KeyedStringDataReader * self,
    struct DDS_KeyedStringSeq * received_data,
    struct 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 )
```

Access a collection of data samples from the **DDS_DataReader** (p. 615).

See also

FooDataReader_read_instance (p. 635)

5.124.3.48 DDS_KeyedStringDataReader_take_instance()

```

DDS_ReturnCode_t DDS_KeyedStringDataReader_take_instance (
    DDS_KeyedStringDataReader * self,
    struct DDS_KeyedStringSeq * received_data,
    struct 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 )

```

Access a collection of data samples from the **DDS_DataReader** (p. 615).

See also

FooDataReader_take_instance (p. 636)

5.124.3.49 DDS_KeyedStringDataReader_read_instance_w_condition()

```

DDS_ReturnCode_t DDS_KeyedStringDataReader_read_instance_w_condition (
    DDS_KeyedStringDataReader * self,
    struct DDS_KeyedStringSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    const DDS_InstanceHandle_t * a_handle,
    DDS_ReadCondition * condition )

```

Accesses via **DDS_KeyedStringDataReader_read_instance** (p. 1146) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).

See also

FooDataReader_read_instance_w_condition (p. 638)

5.124.3.50 DDS_KeyedStringDataReader_take_instance_w_condition()

```

DDS_ReturnCode_t DDS_KeyedStringDataReader_take_instance_w_condition (
    DDS_KeyedStringDataReader * self,
    struct DDS_KeyedStringSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    const DDS_InstanceHandle_t * a_handle,
    DDS_ReadCondition * condition )

```

Accesses via **DDS_KeyedStringDataReader_take_instance** (p. 1146) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).

See also

FooDataReader_take_instance_w_condition (p. 639)

5.124.3.51 DDS_KeyedStringDataReader_read_next_instance()

```
DDS_ReturnCode_t DDS_KeyedStringDataReader_read_next_instance (
    DDS_KeyedStringDataReader * self,
    struct DDS_KeyedStringSeq * received_data,
    struct 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 )
```

Access a collection of data samples from the **DDS_DataReader** (p. 615).

See also

FooDataReader_read_next_instance (p. 640)

5.124.3.52 DDS_KeyedStringDataReader_take_next_instance()

```
DDS_ReturnCode_t DDS_KeyedStringDataReader_take_next_instance (
    DDS_KeyedStringDataReader * self,
    struct DDS_KeyedStringSeq * received_data,
    struct 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 )
```

Access a collection of data samples from the **DDS_DataReader** (p. 615).

See also

FooDataReader_take_next_instance (p. 642)

5.124.3.53 DDS_KeyedStringDataReader_read_next_instance_w_condition()

```
DDS_ReturnCode_t DDS_KeyedStringDataReader_read_next_instance_w_condition (
    DDS_KeyedStringDataReader * self,
    struct DDS_KeyedStringSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    const DDS_InstanceHandle_t * previous_handle,
    DDS_ReadCondition * condition )
```

Accesses via **DDS_KeyedStringDataReader_read_next_instance** (p. 1147) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).

See also

FooDataReader_read_next_instance_w_condition (p. 643)

5.124.3.54 DDS_KeyedStringDataReader_take_next_instance_w_condition()

```
DDS_ReturnCode_t DDS_KeyedStringDataReader_take_next_instance_w_condition (
    DDS_KeyedStringDataReader * self,
    struct DDS_KeyedStringSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    const DDS_InstanceHandle_t * previous_handle,
    DDS_ReadCondition * condition )
```

Accesses via **DDS_KeyedStringDataReader_take_next_instance** (p. 1148) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).

See also

FooDataReader_take_next_instance_w_condition (p. 645)

5.124.3.55 DDS_KeyedStringDataReader_return_loan()

```
DDS_ReturnCode_t DDS_KeyedStringDataReader_return_loan (
    DDS_KeyedStringDataReader * self,
    struct DDS_KeyedStringSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq )
```

Indicates to the **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615).

See also

FooDataReader_return_loan (p. 646)

5.124.3.56 DDS_KeyedStringDataReader_get_key_value()

```
DDS_ReturnCode_t DDS_KeyedStringDataReader_get_key_value (
    DDS_KeyedStringDataReader * self,
    DDS_KeyedString * key_holder,
    const DDS_InstanceHandle_t * handle )
```

Retrieve the instance `key` that corresponds to an instance `handle`.

See also

FooDataReader_get_key_value (p. 647)

5.124.3.57 DDS_KeyedStringDataReader_get_key_value_w_key()

```
DDS_ReturnCode_t DDS_KeyedStringDataReader_get_key_value_w_key (
    DDS_KeyedStringDataReader * self,
    char * key,
    const DDS_InstanceHandle_t * handle )
```

<<*extension*>> (p. 1012) Retrieve the instance `key` that corresponds to an instance `handle`.

See also

FooDataReader_get_key_value (p. 647)

5.124.3.58 DDS_KeyedStringDataReader_lookup_instance()

```
DDS_InstanceHandle_t DDS_KeyedStringDataReader_lookup_instance (
    DDS_KeyedStringDataReader * self,
    const DDS_KeyedString * key_holder )
```

Retrieve the instance `handle` that corresponds to an instance `key_holder`.

See also

FooDataReader_lookup_instance (p. 648)

5.124.3.59 DDS_KeyedStringDataReader_lookup_instance_w_key()

```
DDS_InstanceHandle_t DDS_KeyedStringDataReader_lookup_instance_w_key (
    DDS_KeyedStringDataReader * self,
    const char * key )
```

<<*extension*>> (p. 1012) Retrieve the instance `handle` that corresponds to an instance `key`.

See also

FooDataReader_lookup_instance (p. 648)

5.124.4 Variable Documentation

5.124.4.1 key

```
char* DDS_KeyedString::key
```

Instance key associated with the specified value.

5.124.4.2 value

```
char* DDS_KeyedString::value
```

String value.

5.125 Octets Built-in Type

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

Data Structures

- struct **DDS_Octets**
Built-in type consisting of a variable-length array of opaque bytes.
- struct **DDS_OctetsSeq**
*Instantiates **FooSeq** (p. 2113) < **DDS_Octets** (p. 1859) > .*
- struct **DDS_OctetsTypeSupport**
*<<interface>> (p. 1013) **DDS_Octets** (p. 1859) type support.*

Typedefs

- typedef struct **DDS_Octets DDS_Octets**
Built-in type consisting of a variable-length array of opaque bytes.
- typedef struct **DDS_OctetsDataWriter DDS_OctetsDataWriter**
*<<interface>> (p. 1013) Instantiates *DataWriter* < **DDS_Octets** (p. 1859) > .*
- typedef struct **DDS_OctetsDataReader DDS_OctetsDataReader**
*<<interface>> (p. 1013) Instantiates *DataReader* < **DDS_Octets** (p. 1859) > .*

Functions

- **DDS_Octets * DDS_Octets_new** (void)
Constructor.
- **DDS_Octets * DDS_Octets_new_w_size** (int size)
Constructor that specifies the size of the allocated octets array.
- **void DDS_Octets_delete** (DDS_Octets *self)
Destructor.
- **DDS_ReturnCode_t DDS_OctetsTypeSupport_register_type** (DDS_DomainParticipant *participant, const char *type_name)
*Allows an application to communicate to RTI Connexx the existence of the **DDS_Octets** (p. 1859) data type.*
- **DDS_ReturnCode_t DDS_OctetsTypeSupport_unregister_type** (DDS_DomainParticipant *participant, const char *type_name)
*Allows an application to unregister the **DDS_Octets** (p. 1859) data type from RTI Connexx. After calling unregister_type, no further communication using this type is possible.*
- **const char * DDS_OctetsTypeSupport_get_type_name** (void)
*Get the default name for the **DDS_Octets** (p. 1859) type.*
- **void DDS_OctetsTypeSupport_print_data** (const DDS_Octets *a_data)
<<extension>> (p. 1012) Print value of data type to standard out.
- **DDS_TypeCode * DDS_OctetsTypeSupport_get_typecode** (void)
<<extension>> (p. 1012) Retrieves the TypeCode for the Type.
- **DDS_ReturnCode_t DDS_OctetsTypeSupport_serialize_data_to_cdr_buffer** (char *buffer, unsigned int *length, const DDS_Octets *a_data)
<<extension>> (p. 1012) Serializes the input sample into a CDR buffer of octets.
- **DDS_ReturnCode_t DDS_OctetsTypeSupport_serialize_data_to_cdr_buffer_ex** (char *buffer, unsigned int *length, const DDS_Octets *a_data, DDS_DataRepresentationId_t representation)
<<extension>> (p. 1012) Serializes the input sample into a buffer of octets.
- **DDS_ReturnCode_t DDS_OctetsTypeSupport_deserialize_data_from_cdr_buffer** (DDS_Octets *a_data, const char *buffer, unsigned int length)
<<extension>> (p. 1012) Deserializes a sample from a buffer of octets.
- **DDS_ReturnCode_t DDS_OctetsTypeSupport_data_to_string** (const DDS_Octets *sample, char *str, DDS_UnsignedLong *str_size, DDS_PrintFormatProperty *property)
<<extension>> (p. 1012) Get the string representation of an input sample.
- **DDS_OctetsDataWriter * DDS_OctetsDataWriter_narrow** (DDS_DataWriter *writer)
*Narrow the given **DDS_DataWriter** (p. 485) pointer to a **DDS_OctetsDataWriter** (p. 1154) pointer.*
- **DDS_DataWriter * DDS_OctetsDataWriter_as_datawriter** (DDS_OctetsDataWriter *writer)
*Widen the given **DDS_OctetsDataWriter** (p. 1154) pointer to a **DDS_DataWriter** (p. 485) pointer.*
- **DDS_Octets * DDS_OctetsDataWriter_create_data** (DDS_OctetsDataWriter *self)
Creates an octet data sequence.
- **DDS_Boolean DDS_OctetsDataWriter_delete_data** (DDS_OctetsDataWriter *self, DDS_Octets *sample)
*Destroys a octet data sequence created by **DDS_OctetsDataWriter_create_data** (p. 1161).*
- **DDS_ReturnCode_t DDS_OctetsDataWriter_write** (DDS_OctetsDataWriter *self, const DDS_Octets *instance_data, const DDS_InstanceHandle_t *handle)
*Modifies the value of a **DDS_Octets** (p. 1859) data instance.*
- **DDS_ReturnCode_t DDS_OctetsDataWriter_write_octets** (DDS_OctetsDataWriter *self, const unsigned char *octets, int length, const DDS_InstanceHandle_t *handle)
*<<extension>> (p. 1012) Modifies the value of a **DDS_Octets** (p. 1859) data instance.*
- **DDS_ReturnCode_t DDS_OctetsDataWriter_write_octets_seq** (DDS_OctetsDataWriter *self, const struct DDS_OctetSeq *octets, const DDS_InstanceHandle_t *handle)

<<extension>> (p. 1012) Modifies the value of a **DDS_Octets** (p. 1859) data instance.

- **DDS_ReturnCode_t DDS_OctetsDataWriter_write_w_timestamp** (**DDS_OctetsDataWriter** *self, const **DDS_Octets** *instance_data, const **DDS_InstanceHandle_t** *handle, const struct **DDS_Time_t** *source_↵ timestamp)

Performs the same function as **DDS_OctetsDataWriter_write** (p. 1161) except that it also provides the value for the *source_timestamp*.

- **DDS_ReturnCode_t DDS_OctetsDataWriter_write_octets_w_timestamp** (**DDS_OctetsDataWriter** *self, const unsigned char *octets, int length, const **DDS_InstanceHandle_t** *handle, const struct **DDS_Time_t** *source_timestamp)

<<extension>> (p. 1012) Performs the same function as **DDS_OctetsDataWriter_write_octets** (p. 1162) except that it also provides the value for the *source_timestamp*.

- **DDS_ReturnCode_t DDS_OctetsDataWriter_write_octets_seq_w_timestamp** (**DDS_OctetsDataWriter** *self, const struct **DDS_OctetSeq** *octets, const **DDS_InstanceHandle_t** *handle, const struct **DDS_Time_t** *source_timestamp)

<<extension>> (p. 1012) Performs the same function as **DDS_OctetsDataWriter_write_octets_seq** (p. 1162) except that it also provides the value for the *source_timestamp*.

- **DDS_ReturnCode_t DDS_OctetsDataWriter_write_w_params** (**DDS_OctetsDataWriter** *self, const **DDS_↵ _Octets** *instance_data, struct **DDS_WriteParams_t** *params)

Performs the same function as **DDS_OctetsDataWriter_write** (p. 1161) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

- **DDS_ReturnCode_t DDS_OctetsDataWriter_write_octets_w_params** (**DDS_OctetsDataWriter** *self, const unsigned char *octets, int length, struct **DDS_WriteParams_t** *params)

<<extension>> (p. 1012) Performs the same function as **DDS_OctetsDataWriter_write_octets** (p. 1162) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

- **DDS_ReturnCode_t DDS_OctetsDataWriter_write_octets_seq_w_params** (**DDS_OctetsDataWriter** *self, const struct **DDS_OctetSeq** *octets, struct **DDS_WriteParams_t** *params)

<<extension>> (p. 1012) Performs the same function as **DDS_OctetsDataWriter_write_octets_seq** (p. 1162) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

- **DDS_OctetsDataReader * DDS_OctetsDataReader_narrow** (**DDS_DataReader** *reader)

Narrow the given **DDS_DataReader** (p. 615) pointer to a **DDS_OctetsDataReader** (p. 1154) pointer.

- **DDS_DataReader * DDS_OctetsDataReader_as_datareader** (**DDS_OctetsDataReader** *reader)

Widen the given **DDS_OctetsDataReader** (p. 1154) pointer to a **DDS_DataReader** (p. 615) pointer.

- **DDS_ReturnCode_t DDS_OctetsDataReader_read** (**DDS_OctetsDataReader** *self, struct **DDS_OctetsSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, **DDS_SampleStateMask** sample_states, **DDS_ViewStateMask** view_states, **DDS_InstanceStateMask** instance_states)

Access a collection of data samples from the **DDS_DataReader** (p. 615).

- **DDS_ReturnCode_t DDS_OctetsDataReader_take** (**DDS_OctetsDataReader** *self, struct **DDS_OctetsSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, **DDS_SampleStateMask** sample_mask, **DDS_ViewStateMask** view_mask, **DDS_InstanceStateMask** instance_mask)

Access a collection of data-samples from the **DDS_DataReader** (p. 615).

- **DDS_ReturnCode_t DDS_OctetsDataReader_read_w_condition** (**DDS_OctetsDataReader** *self, struct **DDS_OctetsSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, **DDS_↵ ReadCondition** *condition)

Accesses via **DDS_OctetsDataReader_read** (p. 1166) the samples that match the criteria specified in the **DDS_Read_↵ Condition** (p. 694).

- **DDS_ReturnCode_t DDS_OctetsDataReader_take_w_condition** (**DDS_OctetsDataReader** *self, struct **DDS_OctetsSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, **DDS_↵ ReadCondition** *condition)

Analogous to **DDS_OctetsDataReader_read_w_condition** (p. 1167) except it accesses samples via the **DDS_Octets_↵ DataReader_take** (p. 1166) operation.

- **DDS_ReturnCode_t DDS_OctetsDataReader_read_next_sample** (**DDS_OctetsDataReader** *self, **DDS_Octets** *received_data, struct **DDS_SampleInfo** *sample_info)
*Copies the next not-previously-accessed data value from the **DDS_DataReader** (p. 615).*
- **DDS_ReturnCode_t DDS_OctetsDataReader_take_next_sample** (**DDS_OctetsDataReader** *self, **DDS_Octets** *received_data, struct **DDS_SampleInfo** *sample_info)
*Copies the next not-previously-accessed data value from the **DDS_DataReader** (p. 615).*
- **DDS_ReturnCode_t DDS_OctetsDataReader_return_loan** (**DDS_OctetsDataReader** *self, struct **DDS_OctetsSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq)
*Indicates to the **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615).*

Variables

- int **DDS_Octets::length**
Number of octets to serialize.
- unsigned char * **DDS_Octets::value**
***DDS_Octets** (p. 1859) array value.*

5.125.1 Detailed Description

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

5.125.2 Typedef Documentation

5.125.2.1 DDS_Octets

```
typedef struct DDS_Octets DDS_Octets
```

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

5.125.2.2 DDS_OctetsDataWriter

```
typedef struct DDS_OctetsDataWriter DDS_OctetsDataWriter
```

<<**interface**>> (p. 1013) Instantiates **DataWriter** < **DDS_Octets** (p. 1859) >.

See also

FooDataWriter (p. 2113)

DDS_DataWriter (p. 485)

5.125.2.3 DDS_OctetsDataReader

```
typedef struct DDS_OctetsDataReader DDS_OctetsDataReader
```

<<*interface*>> (p. 1013) Instantiates DataReader < **DDS_Octets** (p. 1859) >.

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. 1502); 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. 2112)

DDS_DataReader (p. 615)

5.125.3 Function Documentation

5.125.3.1 DDS_Octets_new()

```
DDS_Octets * DDS_Octets_new (
    void )
```

Constructor.

The default constructor initializes the newly created object with NULL value, and zero length.

Returns

A new **DDS_Octets** (p. 1859) or NULL if failure.

5.125.3.2 DDS_Octets_new_w_size()

```
DDS_Octets * DDS_Octets_new_w_size (
    int size )
```

Constructor that specifies the size of the allocated octets array.

After this function is called, length is set to zero.

Parameters

<i>size</i>	<< <i>in</i> >> (p. 1013) Size of the allocated octets array Cannot be smaller than zero.

Returns

A new **DDS_Octets** (p. 1859) or NULL if failure.

5.125.3.3 DDS_Octets_delete()

```
void DDS_Octets_delete (
    DDS_Octets * self )
```

Destructor.

5.125.3.4 DDS_OctetsTypeSupport_register_type()

```
DDS_ReturnCode_t DDS_OctetsTypeSupport_register_type (
    DDS_DomainParticipant * participant,
    const char * type_name )
```

Allows an application to communicate to RTI Connext the existence of the **DDS_Octets** (p. 1859) data type.

By default, The **DDS_Octets** (p. 1859) built-in type is automatically registered when a DomainParticipant is created using the type_name returned by **DDS_OctetsTypeSupport_get_type_name** (p. 1158). 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 function can also be used to register the same **DDS_OctetsTypeSupport** (p. 1860) with a **DDS_DomainParticipant** (p. 77) using different values for the type_name.

If **register_type** is called multiple times with the same **DDS_DomainParticipant** (p. 77) and **type_name**, the second (and subsequent) registrations are ignored by the operation.

Parameters

<i>participant</i>	<< <i>in</i> >> (p. 1013) the DDS_DomainParticipant (p. 77) to register the data type DDS_Octets (p. 1859) with. Cannot be NULL.
<i>type_name</i>	<< <i>in</i> >> (p. 1013) the type name under with the data type DDS_Octets (p. 1859) is registered with the participant; this type name is used when creating a new DDS_Topic (p. 175). (See DDS_DomainParticipant_create_topic (p. 115).) The name may not be NULL or longer than 255 characters.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

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

DDS_DomainParticipant_create_topic (p. 115)

5.125.3.5 DDS_OctetsTypeSupport_unregister_type()

```
DDS_ReturnCode_t DDS_OctetsTypeSupport_unregister_type (
    DDS_DomainParticipant * participant,
    const char * type_name )
```

Allows an application to unregister the **DDS_Octets** (p. 1859) data type from RTI Connext. After calling `unregister_type`, no further communication using this type is possible.

Precondition

The **DDS_Octets** (p. 1859) type with `type_name` is registered with the participant and all **DDS_Topic** (p. 175) objects referencing the type have been destroyed. If the type is not registered with the participant, or if any **DDS_Topic** (p. 175) is associated with the type, the operation will fail with **DDS_RETCODE_ERROR** (p. 1222).

Postcondition

All information about the type is removed from RTI Connext. No further communication using this type is possible.

Parameters

<i>participant</i>	<< <i>in</i> >> (p. 1013) the DDS_DomainParticipant (p. 77) to unregister the data type DDS_Octets (p. 1859) from. Cannot be NULL.
<i>type_name</i>	<< <i>in</i> >> (p. 1013) the type name under with the data type DDS_Octets (p. 1859) 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.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_BAD_PARAMETER** (p. 1222) or **DDS_RETCODE_ERROR** (p. 1222)

MT Safety:

SAFE.

See also

DDS_OctetsTypeSupport_register_type (p. 1156)

5.125.3.6 DDS_OctetsTypeSupport_get_type_name()

```
const char * DDS_OctetsTypeSupport_get_type_name (
    void )
```

Get the default name for the **DDS_Octets** (p. 1859) type.

Can be used for calling **DDS_OctetsTypeSupport_register_type** (p. 1156) or creating **DDS_Topic** (p. 175).

Returns

default name for the **DDS_Octets** (p. 1859) type.

See also

DDS_OctetsTypeSupport_register_type (p. 1156)

DDS_DomainParticipant_create_topic (p. 115)

5.125.3.7 DDS_OctetsTypeSupport_print_data()

```
void DDS_OctetsTypeSupport_print_data (
    const DDS_Octets * a_data )
```

<<**extension**>> (p. 1012) Print value of data type to standard out.

The *generated* implementation of the operation knows how to print value of a data type.

Parameters

<i>a_data</i>	<< in >> (p. 1013) DDS_Octets (p. 1859) to be printed.

5.125.3.8 DDS_OctetsTypeSupport_get_typecode()

```
DDS_TypeCode * DDS_OctetsTypeSupport_get_typecode (
    void )
```

<<*extension*>> (p. 1012) Retrieves the TypeCode for the Type.

See also

FooTypeSupport_get_typecode (p. 226)

5.125.3.9 DDS_OctetsTypeSupport_serialize_data_to_cdr_buffer()

```
DDS_ReturnCode_t DDS_OctetsTypeSupport_serialize_data_to_cdr_buffer (
    char * buffer,
    unsigned int * length,
    const DDS_Octets * a_data )
```

<<*extension*>> (p. 1012) Serializes the input sample into a CDR buffer of octets.

See also

FooTypeSupport_serialize_data_to_cdr_buffer (p. 223)

5.125.3.10 DDS_OctetsTypeSupport_serialize_data_to_cdr_buffer_ex()

```
DDS_ReturnCode_t DDS_OctetsTypeSupport_serialize_data_to_cdr_buffer_ex (
    char * buffer,
    unsigned int * length,
    const DDS_Octets * a_data,
    DDS_DataRepresentationId_t representation )
```

<<*extension*>> (p. 1012) Serializes the input sample into a buffer of octets.

See also

FooTypeSupport_serialize_data_to_cdr_buffer (p. 223)

5.125.3.11 DDS_OctetsTypeSupport_deserialize_data_from_cdr_buffer()

```
DDS_ReturnCode_t DDS_OctetsTypeSupport_deserialize_data_from_cdr_buffer (
    DDS_Octets * a_data,
    const char * buffer,
    unsigned int length )
```

<<**extension**>> (p. 1012) Deserializes a sample from a buffer of octets.

See also

FooTypeSupport_deserialize_data_from_cdr_buffer (p. 225)

5.125.3.12 DDS_OctetsTypeSupport_data_to_string()

```
DDS_ReturnCode_t DDS_OctetsTypeSupport_data_to_string (
    const DDS_Octets * sample,
    char * str,
    DDS_UnsignedLong * str_size,
    DDS_PrintFormatProperty * property )
```

<<**extension**>> (p. 1012) Get the string representation of an input sample.

See also

FooTypeSupport_data_to_string (p. 225)

5.125.3.13 DDS_OctetsDataWriter_narrow()

```
DDS_OctetsDataWriter * DDS_OctetsDataWriter_narrow (
    DDS_DataWriter * writer )
```

Narrow the given **DDS_DataWriter** (p. 485) pointer to a **DDS_OctetsDataWriter** (p. 1154) pointer.

See also

FooDataWriter_narrow (p. 490)

5.125.3.14 DDS_OctetsDataWriter_as_datawriter()

```
DDS_DataWriter * DDS_OctetsDataWriter_as_datawriter (
    DDS_OctetsDataWriter * writer )
```

Widen the given **DDS_OctetsDataWriter** (p. 1154) pointer to a **DDS_DataWriter** (p. 485) pointer.

See also

FooDataWriter_as_datawriter (p. 490)

5.125.3.15 DDS_OctetsDataWriter_create_data()

```
DDS_Octets * DDS_OctetsDataWriter_create_data (
    DDS_OctetsDataWriter * self )
```

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 **DDS_OctetsDataWriter_delete_data** (p. 1161).

Returns

Newly created octet sequence data, or NULL on failure.

See also

BuiltinTypeMemoryManagement section of Built-in Types.

5.125.3.16 DDS_OctetsDataWriter_delete_data()

```
DDS_Boolean DDS_OctetsDataWriter_delete_data (
    DDS_OctetsDataWriter * self,
    DDS_Octets * sample )
```

Destroys a octet data sequence created by **DDS_OctetsDataWriter_create_data** (p. 1161).

Returns

DDS_BOOLEAN_TRUE (p. 1201) upon successful deletion.

5.125.3.17 DDS_OctetsDataWriter_write()

```
DDS_ReturnCode_t DDS_OctetsDataWriter_write (
    DDS_OctetsDataWriter * self,
    const DDS_Octets * instance_data,
    const DDS_InstanceHandle_t * handle )
```

Modifies the value of a **DDS_Octets** (p. 1859) data instance.

See also

FooDataWriter_write (p. 496)

5.125.3.18 DDS_OctetsDataWriter_write_octets()

```
DDS_ReturnCode_t DDS_OctetsDataWriter_write_octets (
    DDS_OctetsDataWriter * self,
    const unsigned char * octets,
    int length,
    const DDS_InstanceHandle_t * handle )
```

<<**extension**>> (p. 1012) Modifies the value of a **DDS_Octets** (p. 1859) data instance.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>octets</i>	<< in >> (p. 1013) Array of octets to be published.
<i>length</i>	<< in >> (p. 1013) Number of octets to be published.
<i>handle</i>	<< in >> (p. 1013) The special value DDS_HANDLE_NIL (p. 228) should be used always.

See also

FooDataWriter_write (p. 496)

5.125.3.19 DDS_OctetsDataWriter_write_octets_seq()

```
DDS_ReturnCode_t DDS_OctetsDataWriter_write_octets_seq (
    DDS_OctetsDataWriter * self,
    const struct DDS_OctetSeq * octets,
    const DDS_InstanceHandle_t * handle )
```

<<**extension**>> (p. 1012) Modifies the value of a **DDS_Octets** (p. 1859) data instance.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>octets</i>	<< <i>in</i> >> (p. 1013) Sequence of octets to be published.
<i>handle</i>	<< <i>in</i> >> (p. 1013) The special value DDS_HANDLE_NIL (p. 228) should be used always.

See also

FooDataWriter_write (p. 496)

5.125.3.20 DDS_OctetsDataWriter_write_w_timestamp()

```
DDS_ReturnCode_t DDS_OctetsDataWriter_write_w_timestamp (
    DDS_OctetsDataWriter * self,
    const DDS_Octets * instance_data,
    const DDS_InstanceHandle_t * handle,
    const struct DDS_Time_t * source_timestamp )
```

Performs the same function as **DDS_OctetsDataWriter_write** (p. 1161) except that it also provides the value for the `source_timestamp`.

See also

FooDataWriter_write_w_timestamp (p. 500)

5.125.3.21 DDS_OctetsDataWriter_write_octets_w_timestamp()

```
DDS_ReturnCode_t DDS_OctetsDataWriter_write_octets_w_timestamp (
    DDS_OctetsDataWriter * self,
    const unsigned char * octets,
    int length,
    const DDS_InstanceHandle_t * handle,
    const struct DDS_Time_t * source_timestamp )
```

<<*extension*>> (p. 1012) Performs the same function as **DDS_OctetsDataWriter_write_octets** (p. 1162) except that it also provides the value for the `source_timestamp`.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>octets</i>	<< <i>in</i> >> (p. 1013) Array of octets to be published.
<i>length</i>	<< <i>in</i> >> (p. 1013) Number of octets to be published.
<i>handle</i>	<< <i>in</i> >> (p. 1013) The special value DDS_HANDLE_NIL (p. 228) should be used always.
<i>source_timestamp</i>	<< <i>in</i> >> (p. 1013) 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. 500). Cannot be NULL.

See also

FooDataWriter_write_w_timestamp (p. 500)

5.125.3.22 DDS_OctetsDataWriter_write_octets_seq_w_timestamp()

```
DDS_ReturnCode_t DDS_OctetsDataWriter_write_octets_seq_w_timestamp (
    DDS_OctetsDataWriter * self,
    const struct DDS_OctetSeq * octets,
    const DDS_InstanceHandle_t * handle,
    const struct DDS_Time_t * source_timestamp )
```

<<*extension*>> (p. 1012) Performs the same function as **DDS_OctetsDataWriter_write_octets_seq** (p. 1162) except that it also provides the value for the `source_timestamp`.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>octets</i>	<< <i>in</i> >> (p. 1013) Sequence of octets to be published.
<i>handle</i>	<< <i>in</i> >> (p. 1013) The special value DDS_HANDLE_NIL (p. 228) should be used always.
<i>source_timestamp</i>	<< <i>in</i> >> (p. 1013) 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. 500). Cannot be NULL.

See also

FooDataWriter_write_w_timestamp (p. 500)

5.125.3.23 DDS_OctetsDataWriter_write_w_params()

```
DDS_ReturnCode_t DDS_OctetsDataWriter_write_w_params (
    DDS_OctetsDataWriter * self,
    const DDS_Octets * instance_data,
    struct DDS_WriteParams_t * params )
```

Performs the same function as **DDS_OctetsDataWriter_write** (p. 1161) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

See also

FooDataWriter_write_w_params (p. 501)

5.125.3.24 DDS_OctetsDataWriter_write_octets_w_params()

```
DDS_ReturnCode_t DDS_OctetsDataWriter_write_octets_w_params (
    DDS_OctetsDataWriter * self,
    const unsigned char * octets,
    int length,
    struct DDS_WriteParams_t * params )
```

<<**extension**>> (p. 1012) Performs the same function as **DDS_OctetsDataWriter_write_octets** (p. 1162) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>octets</i>	<< in >> (p. 1013) Array of octets to be published.
<i>length</i>	<< in >> (p. 1013) Number of octets to be published.
<i>params</i>	<< in >> (p. 1013) The DDS_WriteParams_t (p. 2101) parameter containing the instance handle, source timestamp, publication priority, and cookie to be used in write operation. See FooDataWriter_write_w_params (p. 501). Cannot be NULL.

See also

FooDataWriter_write_w_params (p. 501)

5.125.3.25 DDS_OctetsDataWriter_write_octets_seq_w_params()

```
DDS_ReturnCode_t DDS_OctetsDataWriter_write_octets_seq_w_params (
    DDS_OctetsDataWriter * self,
    const struct DDS_OctetSeq * octets,
    struct DDS_WriteParams_t * params )
```

<<**extension**>> (p. 1012) Performs the same function as **DDS_OctetsDataWriter_write_octets_seq** (p. 1162) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>octets</i>	<< in >> (p. 1013) Sequence of octets to be published.
<i>params</i>	<< in >> (p. 1013) The DDS_WriteParams_t (p. 2101) parameter containing the instance handle, source timestamp, publication priority, and cookie to be used in write operation. See FooDataWriter_write_w_params (p. 501). Cannot be NULL.

See also

FooDataWriter_write_w_params (p. 501)

5.125.3.26 DDS_OctetsDataReader_narrow()

```
DDS_OctetsDataReader * DDS_OctetsDataReader_narrow (
    DDS_DataReader * reader )
```

Narrow the given **DDS_DataReader** (p. 615) pointer to a **DDS_OctetsDataReader** (p. 1154) pointer.

See also

FooDataReader_narrow (p. 624)

5.125.3.27 DDS_OctetsDataReader_as_datareader()

```
DDS_DataReader * DDS_OctetsDataReader_as_datareader (
    DDS_OctetsDataReader * reader )
```

Widen the given **DDS_OctetsDataReader** (p. 1154) pointer to a **DDS_DataReader** (p. 615) pointer.

See also

FooDataReader_as_datareader (p. 624)

5.125.3.28 DDS_OctetsDataReader_read()

```
DDS_ReturnCode_t DDS_OctetsDataReader_read (
    DDS_OctetsDataReader * self,
    struct DDS_OctetsSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    DDS_SampleStateMask sample_states,
    DDS_ViewStateMask view_states,
    DDS_InstanceStateMask instance_states )
```

Access a collection of data samples from the **DDS_DataReader** (p. 615).

See also

FooDataReader_read (p. 625)

5.125.3.29 DDS_OctetsDataReader_take()

```

DDS_ReturnCode_t DDS_OctetsDataReader_take (
    DDS_OctetsDataReader * self,
    struct DDS_OctetsSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    DDS_SampleStateMask sample_mask,
    DDS_ViewStateMask view_mask,
    DDS_InstanceStateMask instance_mask )

```

Access a collection of data-samples from the **DDS_DataReader** (p. 615).

See also

FooDataReader_take (p. 626)

5.125.3.30 DDS_OctetsDataReader_read_w_condition()

```

DDS_ReturnCode_t DDS_OctetsDataReader_read_w_condition (
    DDS_OctetsDataReader * self,
    struct DDS_OctetsSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    DDS_ReadCondition * condition )

```

Accesses via **DDS_OctetsDataReader_read** (p. 1166) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).

See also

FooDataReader_read_w_condition (p. 630)

5.125.3.31 DDS_OctetsDataReader_take_w_condition()

```

DDS_ReturnCode_t DDS_OctetsDataReader_take_w_condition (
    DDS_OctetsDataReader * self,
    struct DDS_OctetsSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    DDS_ReadCondition * condition )

```

Analogous to **DDS_OctetsDataReader_read_w_condition** (p. 1167) except it accesses samples via the **DDS_OctetsDataReader_take** (p. 1166) operation.

See also

FooDataReader_take_w_condition (p. 632)

5.125.3.32 DDS_OctetsDataReader_read_next_sample()

```
DDS_ReturnCode_t DDS_OctetsDataReader_read_next_sample (
    DDS_OctetsDataReader * self,
    DDS_Octets * received_data,
    struct DDS_SampleInfo * sample_info )
```

Copies the next not-previously-accessed data value from the **DDS_DataReader** (p. 615).

See also

FooDataReader_read_next_sample (p. 633)

5.125.3.33 DDS_OctetsDataReader_take_next_sample()

```
DDS_ReturnCode_t DDS_OctetsDataReader_take_next_sample (
    DDS_OctetsDataReader * self,
    DDS_Octets * received_data,
    struct DDS_SampleInfo * sample_info )
```

Copies the next not-previously-accessed data value from the **DDS_DataReader** (p. 615).

See also

FooDataReader_take_next_sample (p. 634)

5.125.3.34 DDS_OctetsDataReader_return_loan()

```
DDS_ReturnCode_t DDS_OctetsDataReader_return_loan (
    DDS_OctetsDataReader * self,
    struct DDS_OctetsSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq )
```

Indicates to the **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615).

See also

FooDataReader_return_loan (p. 646)

5.125.4 Variable Documentation

5.125.4.1 length

```
int DDS_Octets::length
```

Number of octets to serialize.

5.125.4.2 value

```
unsigned char* DDS_Octets::value
```

DDS_Octets (p. 1859) array value.

5.126 KeyedOctets Built-in Type

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

Data Structures

- 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. 2113) < **DDS_KeyedOctets** (p. 1811) >.*
- struct **DDS_KeyedOctetsTypeSupport**
*<<interface>> (p. 1013) **DDS_KeyedOctets** (p. 1811) type support.*

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.
- typedef struct **DDS_KeyedOctetsDataWriter DDS_KeyedOctetsDataWriter**
*<<interface>> (p. 1013) Instantiates *DataWriter* < **DDS_KeyedOctets** (p. 1811) >.*
- typedef struct **DDS_KeyedOctetsDataReader DDS_KeyedOctetsDataReader**
*<<interface>> (p. 1013) Instantiates *DataReader* < **DDS_KeyedOctets** (p. 1811) >.*

Functions

- **DDS_KeyedOctets * DDS_KeyedOctets_new** (void)
Constructor.
- **DDS_KeyedOctets * DDS_KeyedOctets_new_w_size** (int key_size, int size)
Constructor that specifies the allocated sizes.
- **void DDS_KeyedOctets_delete** (DDS_KeyedOctets *self)
Destructor.
- **DDS_ReturnCode_t DDS_KeyedOctetsTypeSupport_register_type** (DDS_DomainParticipant *participant, const char *type_name)
*Allows an application to communicate to RTI Connexx the existence of the **DDS_KeyedOctets** (p. 1811) data type.*
- **DDS_ReturnCode_t DDS_KeyedOctetsTypeSupport_unregister_type** (DDS_DomainParticipant *participant, const char *type_name)
*Allows an application to unregister the **DDS_KeyedOctets** (p. 1811) data type from RTI Connexx. After calling unregister_type, no further communication using this type is possible.*
- **const char * DDS_KeyedOctetsTypeSupport_get_type_name** (void)
*Get the default name for the **DDS_KeyedOctets** (p. 1811) type.*
- **void DDS_KeyedOctetsTypeSupport_print_data** (const DDS_KeyedOctets *a_data)
<<extension>> (p. 1012) Print value of data type to standard out.
- **DDS_TypeCode * DDS_KeyedOctetsTypeSupport_get_typecode** (void)
<<extension>> (p. 1012) Retrieves the TypeCode for the Type.
- **DDS_ReturnCode_t DDS_KeyedOctetsTypeSupport_serialize_data_to_cdr_buffer** (char *buffer, unsigned int *length, const DDS_KeyedOctets *a_data)
<<extension>> (p. 1012) Serializes the input sample into a CDR buffer of octets.
- **DDS_ReturnCode_t DDS_KeyedOctetsTypeSupport_serialize_data_to_cdr_buffer_ex** (char *buffer, unsigned int *length, const DDS_KeyedOctets *a_data, DDS_DataRepresentationId_t representation)
<<extension>> (p. 1012) Serializes the input sample into a buffer of octets.
- **DDS_ReturnCode_t DDS_KeyedOctetsTypeSupport_deserialize_data_from_cdr_buffer** (DDS_KeyedOctets *a_data, const char *buffer, unsigned int length)
<<extension>> (p. 1012) Deserializes a sample from a buffer of octets.
- **DDS_ReturnCode_t DDS_KeyedOctetsTypeSupport_data_to_string** (const DDS_KeyedOctets *sample, char *str, DDS_UnsignedLong *str_size, DDS_PrintFormatProperty *property)
<<extension>> (p. 1012) Get the string representation of an input sample.
- **DDS_KeyedOctetsDataWriter * DDS_KeyedOctetsDataWriter_narrow** (DDS_DataWriter *writer)
*Narrow the given **DDS_DataWriter** (p. 485) pointer to a **DDS_KeyedOctetsDataWriter** (p. 1175) pointer.*
- **DDS_DataWriter * DDS_KeyedOctetsDataWriter_as_datawriter** (DDS_KeyedOctetsDataWriter *writer)
*Widen the given **DDS_KeyedOctetsDataWriter** (p. 1175) pointer to a **DDS_DataWriter** (p. 485) pointer.*
- **DDS_InstanceHandle_t DDS_KeyedOctetsDataWriter_register_instance** (DDS_KeyedOctetsDataWriter *self, const DDS_KeyedOctets *instance_data)
Informs RTI Connexx that the application will be modifying a particular instance.
- **DDS_InstanceHandle_t DDS_KeyedOctetsDataWriter_register_instance_w_key** (DDS_KeyedOctetsDataWriter *self, const char *key)
<<extension>> (p. 1012) Inform RTI Connexx that the application will be modifying a particular instance.
- **DDS_InstanceHandle_t DDS_KeyedOctetsDataWriter_register_instance_w_timestamp** (DDS_KeyedOctetsDataWriter *self, const DDS_KeyedOctets *instance_data, const struct DDS_Time_t *source_timestamp)
*Performs the same functions as **DDS_KeyedOctetsDataWriter_register_instance** (p. 1181) except that the application provides the value for the source_timestamp.*

- **DDS_InstanceHandle_t DDS_KeyedOctetsDataWriter_register_instance_w_key_w_timestamp** (**DDS_↵**
KeyedOctetsDataWriter *self, const char *key, const struct **DDS_Time_t** *source_timestamp)
*<<extension>> (p. 1012) Performs the same functions as **DDS_KeyedOctetsDataWriter_register_instance_w_key** (p. 1182) except that the application provides the value for the *source_timestamp*.*
- **DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_unregister_instance** (**DDS_KeyedOctetsDataWriter** *self, const **DDS_KeyedOctets** *instance_data, const **DDS_InstanceHandle_t** *handle)
*Reverses the action of **DDS_KeyedOctetsDataWriter_register_instance** (p. 1181).*
- **DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_unregister_instance_w_key** (**DDS_KeyedOctets**↵
DataWriter *self, const char *key, const **DDS_InstanceHandle_t** *handle)
*<<extension>> (p. 1012) Reverses the action of **DDS_KeyedOctetsDataWriter_register_instance_w_key** (p. 1182).*
- **DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_unregister_instance_w_timestamp** (**DDS_Keyed**↵
OctetsDataWriter *self, const **DDS_KeyedOctets** *instance_data, const **DDS_InstanceHandle_t** *handle, const struct **DDS_Time_t** *source_timestamp)
*Performs the same function as **DDS_KeyedOctetsDataWriter_unregister_instance** (p. 1183) except that it also provides the value for the *source_timestamp*.*
- **DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_unregister_instance_w_key_w_timestamp** (**DDS_**↵
KeyedOctetsDataWriter *self, const char *key, const **DDS_InstanceHandle_t** *handle, const struct **DDS_**↵
Time_t *source_timestamp)
*<<extension>> (p. 1012) Performs the same function as **DDS_KeyedOctetsDataWriter_unregister_instance_w_**↵
key (p. 1183) except that it also provides the value for the *source_timestamp*.*
- **DDS_KeyedOctets * DDS_KeyedOctetsDataWriter_create_data** (**DDS_KeyedOctetsDataWriter** *self)
Creates a keyed octet sequence.
- **DDS_Boolean DDS_KeyedOctetsDataWriter_delete_data** (**DDS_KeyedOctetsDataWriter** *self, **DDS_**↵
KeyedOctets *sample)
*Destroys a string data instance created by **DDS_StringDataWriter_create_data** (p. 1119).*
- **DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_write** (**DDS_KeyedOctetsDataWriter** *self, const **DDS_KeyedOctets** *instance_data, const **DDS_InstanceHandle_t** *handle)
*Modifies the value of a **DDS_KeyedOctets** (p. 1811) data instance.*
- **DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_write_octets_w_key** (**DDS_KeyedOctetsDataWriter** *self, const char *key, const unsigned char *octets, int length, const **DDS_InstanceHandle_t** *handle)
*<<extension>> (p. 1012) Modifies the value of a **DDS_KeyedOctets** (p. 1811) data instance.*
- **DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_write_octets_seq_w_key** (**DDS_KeyedOctetsData**↵
Writer *self, const char *key, const struct **DDS_OctetSeq** *octets, const **DDS_InstanceHandle_t** *handle)
*<<extension>> (p. 1012) Modifies the value of a **DDS_KeyedOctets** (p. 1811) data instance.*
- **DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_write_w_timestamp** (**DDS_KeyedOctetsDataWriter** *self, const **DDS_KeyedOctets** *instance_data, const **DDS_InstanceHandle_t** *handle, const struct **DDS_**↵
Time_t *source_timestamp)
*Performs the same function as **DDS_KeyedOctetsDataWriter_write** (p. 1185) except that it also provides the value for the *source_timestamp*.*
- **DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_write_octets_w_key_w_timestamp** (**DDS_Keyed**↵
OctetsDataWriter *self, const char *key, const unsigned char *octets, int length, const **DDS_InstanceHandle**↵
_t *handle, const struct **DDS_Time_t** *source_timestamp)
*Performs the same function as **DDS_KeyedOctetsDataWriter_write_octets_w_key** (p. 1185) except that it also provides the value for the *source_timestamp*.*
- **DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_write_octets_seq_w_key_w_timestamp** (**DDS_**↵
KeyedOctetsDataWriter *self, const char *key, const struct **DDS_OctetSeq** *octets, const **DDS_Instance**↵
Handle_t *handle, const struct **DDS_Time_t** *source_timestamp)
*Performs the same function as **DDS_KeyedOctetsDataWriter_write_octets_seq_w_key** (p. 1186) except that it also provides the value for the *source_timestamp*.*
- **DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_write_w_params** (**DDS_KeyedOctetsDataWriter** *self, const **DDS_KeyedOctets** *instance_data, struct **DDS_WriteParams_t** *params)

Performs the same function as **DDS_KeyedOctetsDataWriter_write** (p. 1185) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

- **DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_write_octets_w_key_w_params** (**DDS_KeyedOctetsDataWriter** *self, const char *key, const unsigned char *octets, int length, struct **DDS_WriteParams_t** *params)

Performs the same function as **DDS_KeyedOctetsDataWriter_write_octets_w_key** (p. 1185) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

- **DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_write_octets_seq_w_key_w_params** (**DDS_KeyedOctetsDataWriter** *self, const char *key, const struct **DDS_OctetSeq** *octets, struct **DDS_WriteParams_t** *params)

Performs the same function as **DDS_KeyedOctetsDataWriter_write_octets_seq_w_key** (p. 1186) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

- **DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_dispose** (**DDS_KeyedOctetsDataWriter** *self, const **DDS_KeyedOctets** *instance_data, const **DDS_InstanceHandle_t** *instance_handle)

Requests the middleware to delete the data.

- **DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_dispose_w_key** (**DDS_KeyedOctetsDataWriter** *self, const char *key, const **DDS_InstanceHandle_t** *instance_handle)

<<extension>> (p. 1012) Requests the middleware to delete the data.

- **DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_dispose_w_timestamp** (**DDS_KeyedOctetsDataWriter** *self, const **DDS_KeyedOctets** *instance_data, const **DDS_InstanceHandle_t** *instance_handle, const struct **DDS_Time_t** *source_timestamp)

Performs the same functions as **DDS_KeyedOctetsDataWriter_dispose** (p. 1189) except that the application provides the value for the *source_timestamp* that is made available to **DDS_DataReader** (p. 615) objects by means of the *source_timestamp* attribute inside the **DDS_SampleInfo** (p. 1985).

- **DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_dispose_w_key_w_timestamp** (**DDS_KeyedOctetsDataWriter** *self, const char *key, const **DDS_InstanceHandle_t** *instance_handle, const struct **DDS_Time_t** *source_timestamp)

<<extension>> (p. 1012) Performs the same functions as **DDS_KeyedOctetsDataWriter_dispose_w_key** (p. 1189) except that the application provides the value for the *source_timestamp* that is made available to **DDS_DataReader** (p. 615) objects by means of the *source_timestamp* attribute inside the **DDS_SampleInfo** (p. 1985).

- **DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_get_key_value** (**DDS_KeyedOctetsDataWriter** *self, **DDS_KeyedOctets** *key_holder, const **DDS_InstanceHandle_t** *handle)

Retrieve the instance *key* that corresponds to an instance *handle*.

- **DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_get_key_value_w_key** (**DDS_KeyedOctetsDataWriter** *self, char *key, const **DDS_InstanceHandle_t** *handle)

<<extension>> (p. 1012) Retrieve the instance *key* that corresponds to an instance *handle*.

- **DDS_InstanceHandle_t DDS_KeyedOctetsDataWriter_lookup_instance** (**DDS_KeyedOctetsDataWriter** *self, const **DDS_KeyedOctets** *key_holder)

Retrieve the instance *handle* that corresponds to an instance *key_holder*.

- **DDS_InstanceHandle_t DDS_KeyedOctetsDataWriter_lookup_instance_w_key** (**DDS_KeyedOctetsDataWriter** *self, const char *key)

<<extension>> (p. 1012) Retrieve the instance *handle* that corresponds to an instance *key*.

- **DDS_KeyedOctetsDataReader * DDS_KeyedOctetsDataReader_narrow** (**DDS_DataReader** *reader)

Narrow the given **DDS_DataReader** (p. 615) pointer to a **DDS_KeyedOctetsDataReader** (p. 1175) pointer.

- **DDS_DataReader * DDS_KeyedOctetsDataReader_as_datareader** (**DDS_KeyedOctetsDataReader** *reader)

Widen the given **DDS_KeyedOctetsDataReader** (p. 1175) pointer to a **DDS_DataReader** (p. 615) pointer.

- **DDS_ReturnCode_t DDS_KeyedOctetsDataReader_read** (**DDS_KeyedOctetsDataReader** *self, struct **DDS_KeyedOctetsSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, **DDS_SampleStateMask** sample_states, **DDS_ViewStateMask** view_states, **DDS_InstanceStateMask** instance_states)

Access a collection of data samples from the **DDS_DataReader** (p. 615).

- **DDS_ReturnCode_t DDS_KeyedOctetsDataReader_take** (**DDS_KeyedOctetsDataReader** *self, struct **DDS_KeyedOctetsSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, **DDS_SampleStateMask** sample_mask, **DDS_ViewStateMask** view_mask, **DDS_InstanceStateMask** instance_mask)

*Access a collection of data-samples from the **DDS_DataReader** (p. 615).*

- **DDS_ReturnCode_t DDS_KeyedOctetsDataReader_read_w_condition** (**DDS_KeyedOctetsDataReader** *self, struct **DDS_KeyedOctetsSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, **DDS_ReadCondition** *condition)

*Accesses via **DDS_KeyedOctetsDataReader_read** (p. 1192) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).*

- **DDS_ReturnCode_t DDS_KeyedOctetsDataReader_take_w_condition** (**DDS_KeyedOctetsDataReader** *self, struct **DDS_KeyedOctetsSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, **DDS_ReadCondition** *condition)

*Analogous to **DDS_KeyedOctetsDataReader_read_w_condition** (p. 1193) except it accesses samples via the **DDS_KeyedOctetsDataReader_take** (p. 1193) operation.*

- **DDS_ReturnCode_t DDS_KeyedOctetsDataReader_read_next_sample** (**DDS_KeyedOctetsDataReader** *self, **DDS_KeyedOctets** *received_data, struct **DDS_SampleInfo** *sample_info)

*Copies the next not-previously-accessed data value from the **DDS_DataReader** (p. 615).*

- **DDS_ReturnCode_t DDS_KeyedOctetsDataReader_take_next_sample** (**DDS_KeyedOctetsDataReader** *self, **DDS_KeyedOctets** *received_data, struct **DDS_SampleInfo** *sample_info)

*Copies the next not-previously-accessed data value from the **DDS_DataReader** (p. 615).*

- **DDS_ReturnCode_t DDS_KeyedOctetsDataReader_read_instance** (**DDS_KeyedOctetsDataReader** *self, struct **DDS_KeyedOctetsSeq** *received_data, struct **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)

*Access a collection of data samples from the **DDS_DataReader** (p. 615).*

- **DDS_ReturnCode_t DDS_KeyedOctetsDataReader_take_instance** (**DDS_KeyedOctetsDataReader** *self, struct **DDS_KeyedOctetsSeq** *received_data, struct **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)

*Access a collection of data samples from the **DDS_DataReader** (p. 615).*

- **DDS_ReturnCode_t DDS_KeyedOctetsDataReader_read_instance_w_condition** (**DDS_KeyedOctetsDataReader** *self, struct **DDS_KeyedOctetsSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, const **DDS_InstanceHandle_t** *a_handle, **DDS_ReadCondition** *condition)

*Accesses via **DDS_KeyedOctetsDataReader_read_instance** (p. 1194) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).*

- **DDS_ReturnCode_t DDS_KeyedOctetsDataReader_take_instance_w_condition** (**DDS_KeyedOctetsDataReader** *self, struct **DDS_KeyedOctetsSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, const **DDS_InstanceHandle_t** *a_handle, **DDS_ReadCondition** *condition)

*Accesses via **DDS_KeyedOctetsDataReader_take_instance** (p. 1195) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).*

- **DDS_ReturnCode_t DDS_KeyedOctetsDataReader_read_next_instance** (**DDS_KeyedOctetsDataReader** *self, struct **DDS_KeyedOctetsSeq** *received_data, struct **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)

*Access a collection of data samples from the **DDS_DataReader** (p. 615).*

- **DDS_ReturnCode_t DDS_KeyedOctetsDataReader_take_next_instance** (**DDS_KeyedOctetsDataReader** *self, struct **DDS_KeyedOctetsSeq** *received_data, struct **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)

*Access a collection of data samples from the **DDS_DataReader** (p. 615).*

- **DDS_ReturnCode_t DDS_KeyedOctetsDataReader_read_next_instance_w_condition** (**DDS_KeyedOctetsDataReader** *self, struct **DDS_KeyedOctetsSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, const **DDS_InstanceHandle_t** *previous_handle, **DDS_ReadCondition** *condition)
*Accesses via **DDS_KeyedOctetsDataReader_read_next_instance** (p. 1196) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).*
- **DDS_ReturnCode_t DDS_KeyedOctetsDataReader_take_next_instance_w_condition** (**DDS_KeyedOctetsDataReader** *self, struct **DDS_KeyedOctetsSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq, **DDS_Long** max_samples, const **DDS_InstanceHandle_t** *previous_handle, **DDS_ReadCondition** *condition)
*Accesses via **DDS_KeyedOctetsDataReader_take_next_instance** (p. 1196) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).*
- **DDS_ReturnCode_t DDS_KeyedOctetsDataReader_return_loan** (**DDS_KeyedOctetsDataReader** *self, struct **DDS_KeyedOctetsSeq** *received_data, struct **DDS_SampleInfoSeq** *info_seq)
*Indicates to the **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615).*
- **DDS_ReturnCode_t DDS_KeyedOctetsDataReader_get_key_value** (**DDS_KeyedOctetsDataReader** *self, **DDS_KeyedOctets** *key_holder, const **DDS_InstanceHandle_t** *handle)
*Retrieve the instance *key* that corresponds to an instance *handle*.*
- **DDS_ReturnCode_t DDS_KeyedOctetsDataReader_get_key_value_w_key** (**DDS_KeyedOctetsDataReader** *self, char *key, const **DDS_InstanceHandle_t** *handle)
*<<extension>> (p. 1012) Retrieve the instance *key* that corresponds to an instance *handle*.*
- **DDS_InstanceHandle_t DDS_KeyedOctetsDataReader_lookup_instance** (**DDS_KeyedOctetsDataReader** *self, const **DDS_KeyedOctets** *key_holder)
*Retrieve the instance *handle* that corresponds to an instance *key_holder*.*
- **DDS_InstanceHandle_t DDS_KeyedOctetsDataReader_lookup_instance_w_key** (**DDS_KeyedOctetsDataReader** *self, const char *key)
*<<extension>> (p. 1012) Retrieve the instance *handle* that corresponds to an instance *key*.*

Variables

- char * **DDS_KeyedOctets::key**
Instance key associated with the specified value.
- int **DDS_KeyedOctets::length**
Number of octets to serialize.
- unsigned char * **DDS_KeyedOctets::value**
***DDS_Octets** (p. 1859) array value.*

5.126.1 Detailed Description

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

5.126.2 Typedef Documentation

5.126.2.1 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.

5.126.2.2 DDS_KeyedOctetsDataWriter

```
typedef struct DDS_KeyedOctetsDataWriter DDS_KeyedOctetsDataWriter
```

<<*interface*>> (p. 1013) Instantiates `DataWriter` < **DDS_KeyedOctets** (p. 1811) >.

See also

FooDataWriter (p. 2113)

DDS_DataWriter (p. 485)

5.126.2.3 DDS_KeyedOctetsDataReader

```
typedef struct DDS_KeyedOctetsDataReader DDS_KeyedOctetsDataReader
```

<<*interface*>> (p. 1013) Instantiates `DataReader` < **DDS_KeyedOctets** (p. 1811) >.

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. 1528); new arrays will be allocated with **DDS_OctetBuffer_alloc** (p. 1502). 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. 2112)

DDS_DataReader (p. 615)

5.126.3 Function Documentation

5.126.3.1 DDS_KeyedOctets_new()

```
DDS_KeyedOctets * DDS_KeyedOctets_new (
    void )
```

Constructor.

The default constructor initializes the newly created object with NULL key, NULL value, and zero length.

Returns

A new **DDS_KeyedOctets** (p. 1811) or NULL if failure.

5.126.3.2 DDS_KeyedOctets_new_w_size()

```
DDS_KeyedOctets * DDS_KeyedOctets_new_w_size (
    int key_size,
    int size )
```

Constructor that specifies the allocated sizes.

After this function is called, key is initialized with the empty string and length is set to zero.

Parameters

<i>key_size</i>	<< <i>in</i> >> (p. 1013) Size of the allocated key (with NULL-terminated character). Cannot be smaller than one.
<i>size</i>	<< <i>in</i> >> (p. 1013) Size of the allocated octets array. Cannot be smaller than zero.

Returns

A new **DDS_KeyedOctets** (p. 1811) or NULL if failure.

5.126.3.3 DDS_KeyedOctets_delete()

```
void DDS_KeyedOctets_delete (
    DDS_KeyedOctets * self )
```

Destructor.

5.126.3.4 DDS_KeyedOctetsTypeSupport_register_type()

```
DDS_ReturnCode_t DDS_KeyedOctetsTypeSupport_register_type (
    DDS_DomainParticipant * participant,
    const char * type_name )
```

Allows an application to communicate to RTI Connext the existence of the **DDS_KeyedOctets** (p. 1811) data type.

By default, The **DDS_KeyedOctets** (p. 1811) built-in type is automatically registered when a DomainParticipant is created using the type_name returned by **DDS_KeyedOctetsTypeSupport_get_type_name** (p. 1178). 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 function can also be used to register the same **DDS_KeyedOctetsTypeSupport** (p. 1812) with a **DDS_DomainParticipant** (p. 77) using different values for the type_name.

If register_type is called multiple times with the same **DDS_DomainParticipant** (p. 77) and type_name, the second (and subsequent) registrations are ignored by the operation.

Parameters

<i>participant</i>	<<in>> (p. 1013) the DDS_DomainParticipant (p. 77) to register the data type DDS_Octets (p. 1859) with. Cannot be NULL.
<i>type_name</i>	<<in>> (p. 1013) the type name under with the data type DDS_KeyedOctets (p. 1811) is registered with the participant; this type name is used when creating a new DDS_Topic (p. 175). (See DDS_DomainParticipant_create_topic (p. 115).) The name may not be NULL or longer than 255 characters.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

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

DDS_DomainParticipant_create_topic (p. 115)

5.126.3.5 DDS_KeyedOctetsTypeSupport_unregister_type()

```
DDS_ReturnCode_t DDS_KeyedOctetsTypeSupport_unregister_type (
    DDS_DomainParticipant * participant,
    const char * type_name )
```

Allows an application to unregister the **DDS_KeyedOctets** (p.1811) data type from RTI Connex. After calling `unregister_type`, no further communication using this type is possible.

Precondition

The **DDS_KeyedOctets** (p.1811) type with `type_name` is registered with the participant and all **DDS_Topic** (p.175) objects referencing the type have been destroyed. If the type is not registered with the participant, or if any **DDS_Topic** (p.175) is associated with the type, the operation will fail with **DDS_RETCODE_ERROR** (p.1222).

Postcondition

All information about the type is removed from RTI Connex. No further communication using this type is possible.

Parameters

<i>participant</i>	<< <i>in</i> >> (p.1013) the DDS_DomainParticipant (p.77) to unregister the data type DDS_KeyedOctets (p.1811) from. Cannot be NULL.
<i>type_name</i>	<< <i>in</i> >> (p.1013) the type name under with the data type DDS_KeyedOctets (p.1811) 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.

Returns

One of the **Standard Return Codes** (p.1221), **DDS_RETCODE_BAD_PARAMETER** (p.1222) or **DDS_RETCODE_ERROR** (p.1222)

MT Safety:

SAFE.

See also

DDS_KeyedOctetsTypeSupport_register_type (p.1176)

5.126.3.6 DDS_KeyedOctetsTypeSupport_get_type_name()

```
const char * DDS_KeyedOctetsTypeSupport_get_type_name (
    void )
```

Get the default name for the **DDS_KeyedOctets** (p. 1811) type.

Can be used for calling **DDS_KeyedOctetsTypeSupport_register_type** (p. 1176) or creating **DDS_Topic** (p. 175).

Returns

default name for the **DDS_KeyedOctets** (p. 1811) type.

See also

DDS_KeyedOctetsTypeSupport_register_type (p. 1176)

DDS_DomainParticipant_create_topic (p. 115)

5.126.3.7 DDS_KeyedOctetsTypeSupport_print_data()

```
void DDS_KeyedOctetsTypeSupport_print_data (
    const DDS_KeyedOctets * a_data )
```

<<**extension**>> (p. 1012) Print value of data type to standard out.

The *generated* implementation of the operation knows how to print value of a data type.

Parameters

<i>a_data</i>	<< in >> (p. 1013) DDS_KeyedOctets (p. 1811) to be printed.

5.126.3.8 DDS_KeyedOctetsTypeSupport_get_typecode()

```
DDS_TypeCode * DDS_KeyedOctetsTypeSupport_get_typecode (
    void )
```

<<**extension**>> (p. 1012) Retrieves the TypeCode for the Type.

See also

FooTypeSupport_get_typecode (p. 226)

5.126.3.9 DDS_KeyedOctetsTypeSupport_serialize_data_to_cdr_buffer()

```
DDS_ReturnCode_t DDS_KeyedOctetsTypeSupport_serialize_data_to_cdr_buffer (
    char * buffer,
    unsigned int * length,
    const DDS_KeyedOctets * a_data )
```

<<*extension*>> (p. 1012) Serializes the input sample into a CDR buffer of octets.

See also

FooTypeSupport_serialize_data_to_cdr_buffer (p. 223)

5.126.3.10 DDS_KeyedOctetsTypeSupport_serialize_data_to_cdr_buffer_ex()

```
DDS_ReturnCode_t DDS_KeyedOctetsTypeSupport_serialize_data_to_cdr_buffer_ex (
    char * buffer,
    unsigned int * length,
    const DDS_KeyedOctets * a_data,
    DDS_DataRepresentationId_t representation )
```

<<*extension*>> (p. 1012) Serializes the input sample into a buffer of octets.

See also

FooTypeSupport_serialize_data_to_cdr_buffer (p. 223)

5.126.3.11 DDS_KeyedOctetsTypeSupport_deserialize_data_from_cdr_buffer()

```
DDS_ReturnCode_t DDS_KeyedOctetsTypeSupport_deserialize_data_from_cdr_buffer (
    DDS_KeyedOctets * a_data,
    const char * buffer,
    unsigned int length )
```

<<*extension*>> (p. 1012) Deserializes a sample from a buffer of octets.

See also

FooTypeSupport_deserialize_data_from_cdr_buffer (p. 225)

5.126.3.12 DDS_KeyedOctetsTypeSupport_data_to_string()

```
DDS_ReturnCode_t DDS_KeyedOctetsTypeSupport_data_to_string (
    const DDS_KeyedOctets * sample,
    char * str,
    DDS_UnsignedLong * str_size,
    DDS_PrintFormatProperty * property )
```

<<*extension*>> (p. 1012) Get the string representation of an input sample.

See also

FooTypeSupport_data_to_string (p. 225)

5.126.3.13 DDS_KeyedOctetsDataWriter_narrow()

```
DDS_KeyedOctetsDataWriter * DDS_KeyedOctetsDataWriter_narrow (
    DDS_DataWriter * writer )
```

Narrow the given **DDS_DataWriter** (p. 485) pointer to a **DDS_KeyedOctetsDataWriter** (p. 1175) pointer.

See also

FooDataWriter_narrow (p. 490)

5.126.3.14 DDS_KeyedOctetsDataWriter_as_datawriter()

```
DDS_DataWriter * DDS_KeyedOctetsDataWriter_as_datawriter (
    DDS_KeyedOctetsDataWriter * writer )
```

Widen the given **DDS_KeyedOctetsDataWriter** (p. 1175) pointer to a **DDS_DataWriter** (p. 485) pointer.

See also

FooDataWriter_as_datawriter (p. 490)

5.126.3.15 DDS_KeyedOctetsDataWriter_register_instance()

```
DDS_InstanceHandle_t DDS_KeyedOctetsDataWriter_register_instance (
    DDS_KeyedOctetsDataWriter * self,
    const DDS_KeyedOctets * instance_data )
```

Informs RTI Connexx that the application will be modifying a particular instance.

See also

FooDataWriter_register_instance (p. 491)

5.126.3.16 DDS_KeyedOctetsDataWriter_register_instance_w_key()

```
DDS_InstanceHandle_t DDS_KeyedOctetsDataWriter_register_instance_w_key (
    DDS_KeyedOctetsDataWriter * self,
    const char * key )
```

<<**extension**>> (p. 1012) Informes RTI Connexx that the application will be modifying a particular instance.

See also

FooDataWriter_register_instance (p. 491)

5.126.3.17 DDS_KeyedOctetsDataWriter_register_instance_w_timestamp()

```
DDS_InstanceHandle_t DDS_KeyedOctetsDataWriter_register_instance_w_timestamp (
    DDS_KeyedOctetsDataWriter * self,
    const DDS_KeyedOctets * instance_data,
    const struct DDS_Time_t * source_timestamp )
```

Performs the same functions as **DDS_KeyedOctetsDataWriter_register_instance** (p. 1181) except that the application provides the value for the `source_timestamp`.

See also

FooDataWriter_register_instance_w_timestamp (p. 492)

5.126.3.18 DDS_KeyedOctetsDataWriter_register_instance_w_key_w_timestamp()

```
DDS_InstanceHandle_t DDS_KeyedOctetsDataWriter_register_instance_w_key_w_timestamp (
    DDS_KeyedOctetsDataWriter * self,
    const char * key,
    const struct DDS_Time_t * source_timestamp )
```

<<*extension*>> (p. 1012) Performs the same functions as **DDS_KeyedOctetsDataWriter_register_instance_w_key** (p. 1182) except that the application provides the value for the `source_timestamp`.

See also

FooDataWriter_register_instance_w_timestamp (p. 492)

5.126.3.19 DDS_KeyedOctetsDataWriter_unregister_instance()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_unregister_instance (
    DDS_KeyedOctetsDataWriter * self,
    const DDS_KeyedOctets * instance_data,
    const DDS_InstanceHandle_t * handle )
```

Reverses the action of **DDS_KeyedOctetsDataWriter_register_instance** (p. 1181).

See also

FooDataWriter_unregister_instance (p. 493)

5.126.3.20 DDS_KeyedOctetsDataWriter_unregister_instance_w_key()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_unregister_instance_w_key (
    DDS_KeyedOctetsDataWriter * self,
    const char * key,
    const DDS_InstanceHandle_t * handle )
```

<<*extension*>> (p. 1012) Reverses the action of **DDS_KeyedOctetsDataWriter_register_instance_w_key** (p. 1182).

See also

FooDataWriter_unregister_instance (p. 493)

5.126.3.21 DDS_KeyedOctetsDataWriter_unregister_instance_w_timestamp()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_unregister_instance_w_timestamp (
    DDS_KeyedOctetsDataWriter * self,
    const DDS_KeyedOctets * instance_data,
    const DDS_InstanceHandle_t * handle,
    const struct DDS_Time_t * source_timestamp )
```

Performs the same function as **DDS_KeyedOctetsDataWriter_unregister_instance** (p. 1183) except that it also provides the value for the `source_timestamp`.

See also

FooDataWriter_unregister_instance_w_timestamp (p. 495)

5.126.3.22 DDS_KeyedOctetsDataWriter_unregister_instance_w_key_w_timestamp()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_unregister_instance_w_key_w_timestamp (
    DDS_KeyedOctetsDataWriter * self,
    const char * key,
    const DDS_InstanceHandle_t * handle,
    const struct DDS_Time_t * source_timestamp )
```

<<*extension*>> (p. 1012) Performs the same function as **DDS_KeyedOctetsDataWriter_unregister_instance_w_key** (p. 1183) except that it also provides the value for the `source_timestamp`.

See also

FooDataWriter_unregister_instance_w_timestamp (p. 495)

5.126.3.23 DDS_KeyedOctetsDataWriter_create_data()

```
DDS_KeyedOctets * DDS_KeyedOctetsDataWriter_create_data (
    DDS_KeyedOctetsDataWriter * self )
```

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 **DDS_KeyedOctetsDataWriter_delete_data** (p. 1184).

Returns

Newly created keyed octet sequence, or NULL on failure.

See also

BuiltinTypeMemoryManagement section of Built-in Types.

5.126.3.24 DDS_KeyedOctetsDataWriter_delete_data()

```
DDS_Boolean DDS_KeyedOctetsDataWriter_delete_data (
    DDS_KeyedOctetsDataWriter * self,
    DDS_KeyedOctets * sample )
```

Destroys a string data instance created by **DDS_StringDataWriter_create_data** (p. 1119).

Returns

DDS_BOOLEAN_TRUE (p. 1201) upon successful deletion.

5.126.3.25 DDS_KeyedOctetsDataWriter_write()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_write (
    DDS_KeyedOctetsDataWriter * self,
    const DDS_KeyedOctets * instance_data,
    const DDS_InstanceHandle_t * handle )
```

Modifies the value of a **DDS_KeyedOctets** (p. 1811) data instance.

See also

FooDataWriter_write (p. 496)

5.126.3.26 DDS_KeyedOctetsDataWriter_write_octets_w_key()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_write_octets_w_key (
    DDS_KeyedOctetsDataWriter * self,
    const char * key,
    const unsigned char * octets,
    int length,
    const DDS_InstanceHandle_t * handle )
```

<<**extension**>> (p. 1012) Modifies the value of a **DDS_KeyedOctets** (p. 1811) data instance.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>key</i>	<< in >> (p. 1013) Instance key.
<i>octets</i>	<< in >> (p. 1013) Array of octets to be published.
<i>length</i>	<< in >> (p. 1013) Number of octets to be published.
<i>handle</i>	<< in >> (p. 1013) Either the handle returned by a previous call to DDS_KeyedOctetsDataWriter_register_instance (p. 1181), or else the special value DDS_HANDLE_NIL (p. 228). See FooDataWriter_write (p. 496).
Generated by Doxygen	

See also

FooDataWriter_write (p. 496)

5.126.3.27 DDS_KeyedOctetsDataWriter_write_octets_seq_w_key()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_write_octets_seq_w_key (
    DDS_KeyedOctetsDataWriter * self,
    const char * key,
    const struct DDS_OctetSeq * octets,
    const DDS_InstanceHandle_t * handle )
```

<<**extension**>> (p. 1012) Modifies the value of a **DDS_KeyedOctets** (p. 1811) data instance.

Parameters

<i>self</i>	<< in >> (p. 1013) Cannot be NULL.
<i>key</i>	<< in >> (p. 1013) Instance key.
<i>octets</i>	<< in >> (p. 1013) Sequence of octets to be published.
<i>handle</i>	<< in >> (p. 1013) Either the handle returned by a previous call to DDS_KeyedOctetsDataWriter_register_instance (p. 1181), or else the special value DDS_HANDLE_NIL (p. 228). See FooDataWriter_write (p. 496).

See also

FooDataWriter_write (p. 496)

5.126.3.28 DDS_KeyedOctetsDataWriter_write_w_timestamp()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_write_w_timestamp (
    DDS_KeyedOctetsDataWriter * self,
    const DDS_KeyedOctets * instance_data,
    const DDS_InstanceHandle_t * handle,
    const struct DDS_Time_t * source_timestamp )
```

Performs the same function as **DDS_KeyedOctetsDataWriter_write** (p. 1185) except that it also provides the value for the `source_timestamp`.

See also

FooDataWriter_write_w_timestamp (p. 500)

5.126.3.29 DDS_KeyedOctetsDataWriter_write_octets_w_key_w_timestamp()

```

DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_write_octets_w_key_w_timestamp (
    DDS_KeyedOctetsDataWriter * self,
    const char * key,
    const unsigned char * octets,
    int length,
    const DDS_InstanceHandle_t * handle,
    const struct DDS_Time_t * source_timestamp )

```

Performs the same function as **DDS_KeyedOctetsDataWriter_write_octets_w_key** (p. 1185) except that it also provides the value for the `source_timestamp`.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>key</i>	<< <i>in</i> >> (p. 1013) Instance key.
<i>octets</i>	<< <i>in</i> >> (p. 1013) Array of octets to be published.
<i>length</i>	<< <i>in</i> >> (p. 1013) Number of octets to be published.
<i>handle</i>	<< <i>in</i> >> (p. 1013) Either the handle returned by a previous call to DDS_KeyedOctetsDataWriter_register_instance (p. 1181), or else the special value DDS_HANDLE_NIL (p. 228). See FooDataWriter_write (p. 496).
<i>source_timestamp</i>	<< <i>in</i> >> (p. 1013) 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. 500). Cannot be NULL.

See also

FooDataWriter_write (p. 496)

5.126.3.30 DDS_KeyedOctetsDataWriter_write_octets_seq_w_key_w_timestamp()

```

DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_write_octets_seq_w_key_w_timestamp (
    DDS_KeyedOctetsDataWriter * self,
    const char * key,
    const struct DDS_OctetSeq * octets,
    const DDS_InstanceHandle_t * handle,
    const struct DDS_Time_t * source_timestamp )

```

Performs the same function as **DDS_KeyedOctetsDataWriter_write_octets_seq_w_key** (p. 1186) except that it also provides the value for the `source_timestamp`.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>key</i>	<< <i>in</i> >> (p. 1013) Instance key.
<i>octets</i>	<< <i>in</i> >> (p. 1013) Sequence of octets to be published.

Parameters

<i>handle</i>	<< <i>in</i> >> (p. 1013) Either the handle returned by a previous call to DDS_KeyedOctetsDataWriter_register_instance (p. 1181), or else the special value DDS_HANDLE NIL (p. 228). See FooDataWriter_write (p. 496).
<i>source_timestamp</i>	<< <i>in</i> >> (p. 1013) 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. 500). Cannot be NULL.

See also

FooDataWriter_write (p. 496)

5.126.3.31 DDS_KeyedOctetsDataWriter_write_w_params()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_write_w_params (
    DDS_KeyedOctetsDataWriter * self,
    const DDS_KeyedOctets * instance_data,
    struct DDS_WriteParams_t * params )
```

Performs the same function as **DDS_KeyedOctetsDataWriter_write** (p. 1185) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

See also

FooDataWriter_write_w_params (p. 501)

5.126.3.32 DDS_KeyedOctetsDataWriter_write_octets_w_key_w_params()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_write_octets_w_key_w_params (
    DDS_KeyedOctetsDataWriter * self,
    const char * key,
    const unsigned char * octets,
    int length,
    struct DDS_WriteParams_t * params )
```

Performs the same function as **DDS_KeyedOctetsDataWriter_write_octets_w_key** (p. 1185) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>key</i>	<< <i>in</i> >> (p. 1013) Instance key.
<i>octets</i>	<< <i>in</i> >> (p. 1013) Array of octets to be published.
<i>length</i>	<< <i>in</i> >> (p. 1013) Number of octets to be published. Generated by Doxygen
<i>params</i>	<< <i>in</i> >> (p. 1013) The DDS_WriteParams_t (p. 2101) parameter containing the instance handle, source timestamp, publication priority, and cookie to be used in write operation. See FooDataWriter_write_w_params (p. 501). Cannot be NULL.

See also

FooDataWriter_write_w_params (p. 501)

5.126.3.33 DDS_KeyedOctetsDataWriter_write_octets_seq_w_key_w_params()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_write_octets_seq_w_key_w_params (
    DDS_KeyedOctetsDataWriter * self,
    const char * key,
    const struct DDS_OctetSeq * octets,
    struct DDS_WriteParams_t * params )
```

Performs the same function as **DDS_KeyedOctetsDataWriter_write_octets_seq_w_key** (p. 1186) except that it also allows specification of the instance handle, source timestamp, publication priority, and cookie.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>key</i>	<< <i>in</i> >> (p. 1013) Instance key.
<i>octets</i>	<< <i>in</i> >> (p. 1013) Sequence of octets to be published.
<i>params</i>	<< <i>in</i> >> (p. 1013) The DDS_WriteParams_t (p. 2101) parameter containing the instance handle, source timestamp, publication priority, and cookie to be used in write operation. See FooDataWriter_write_w_params (p. 501). Cannot be NULL.

See also

FooDataWriter_write_w_params (p. 501)

5.126.3.34 DDS_KeyedOctetsDataWriter_dispose()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_dispose (
    DDS_KeyedOctetsDataWriter * self,
    const DDS_KeyedOctets * instance_data,
    const DDS_InstanceHandle_t * instance_handle )
```

Requests the middleware to delete the data.

See also

FooDataWriter_dispose (p. 502)

5.126.3.35 DDS_KeyedOctetsDataWriter_dispose_w_key()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_dispose_w_key (
    DDS_KeyedOctetsDataWriter * self,
    const char * key,
    const DDS_InstanceHandle_t * instance_handle )
```

<<*extension*>> (p. 1012) Requests the middleware to delete the data.

See also

FooDataWriter_dispose (p. 502)

5.126.3.36 DDS_KeyedOctetsDataWriter_dispose_w_timestamp()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_dispose_w_timestamp (
    DDS_KeyedOctetsDataWriter * self,
    const DDS_KeyedOctets * instance_data,
    const DDS_InstanceHandle_t * instance_handle,
    const struct DDS_Time_t * source_timestamp )
```

Performs the same functions as **DDS_KeyedOctetsDataWriter_dispose** (p. 1189) except that the application provides the value for the `source_timestamp` that is made available to **DDS_DataReader** (p. 615) objects by means of the `source_timestamp` attribute inside the **DDS_SampleInfo** (p. 1985).

See also

FooDataWriter_dispose_w_timestamp (p. 503)

5.126.3.37 DDS_KeyedOctetsDataWriter_dispose_w_key_w_timestamp()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_dispose_w_key_w_timestamp (
    DDS_KeyedOctetsDataWriter * self,
    const char * key,
    const DDS_InstanceHandle_t * instance_handle,
    const struct DDS_Time_t * source_timestamp )
```

<<*extension*>> (p. 1012) Performs the same functions as **DDS_KeyedOctetsDataWriter_dispose_w_key** (p. 1189) except that the application provides the value for the `source_timestamp` that is made available to **DDS_DataReader** (p. 615) objects by means of the `source_timestamp` attribute inside the **DDS_SampleInfo** (p. 1985).

See also

FooDataWriter_dispose_w_timestamp (p. 503)

5.126.3.38 DDS_KeyedOctetsDataWriter_get_key_value()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_get_key_value (
    DDS_KeyedOctetsDataWriter * self,
    DDS_KeyedOctets * key_holder,
    const DDS_InstanceHandle_t * handle )
```

Retrieve the instance `key` that corresponds to an instance `handle`.

See also

FooDataWriter_get_key_value (p. 505)

5.126.3.39 DDS_KeyedOctetsDataWriter_get_key_value_w_key()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataWriter_get_key_value_w_key (
    DDS_KeyedOctetsDataWriter * self,
    char * key,
    const DDS_InstanceHandle_t * handle )
```

<<*extension*>> (p. 1012) Retrieve the instance `key` that corresponds to an instance `handle`.

See also

FooDataWriter_get_key_value (p. 505)

5.126.3.40 DDS_KeyedOctetsDataWriter_lookup_instance()

```
DDS_InstanceHandle_t DDS_KeyedOctetsDataWriter_lookup_instance (
    DDS_KeyedOctetsDataWriter * self,
    const DDS_KeyedOctets * key_holder )
```

Retrieve the instance `handle` that corresponds to an instance `key_holder`.

See also

FooDataWriter_lookup_instance (p. 506)

5.126.3.41 DDS_KeyedOctetsDataWriter_lookup_instance_w_key()

```
DDS_InstanceHandle_t DDS_KeyedOctetsDataWriter_lookup_instance_w_key (
    DDS_KeyedOctetsDataWriter * self,
    const char * key )
```

<<*extension*>> (p. 1012) Retrieve the instance handle that corresponds to an instance key.

See also

FooDataWriter_lookup_instance (p. 506)

5.126.3.42 DDS_KeyedOctetsDataReader_narrow()

```
DDS_KeyedOctetsDataReader * DDS_KeyedOctetsDataReader_narrow (
    DDS_DataReader * reader )
```

Narrow the given **DDS_DataReader** (p. 615) pointer to a **DDS_KeyedOctetsDataReader** (p. 1175) pointer.

See also

FooDataReader_narrow (p. 624)

5.126.3.43 DDS_KeyedOctetsDataReader_as_datareader()

```
DDS_DataReader * DDS_KeyedOctetsDataReader_as_datareader (
    DDS_KeyedOctetsDataReader * reader )
```

Widen the given **DDS_KeyedOctetsDataReader** (p. 1175) pointer to a **DDS_DataReader** (p. 615) pointer.

See also

FooDataReader_as_datareader (p. 624)

5.126.3.44 DDS_KeyedOctetsDataReader_read()

```

DDS_ReturnCode_t DDS_KeyedOctetsDataReader_read (
    DDS_KeyedOctetsDataReader * self,
    struct DDS_KeyedOctetsSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    DDS_SampleStateMask sample_states,
    DDS_ViewStateMask view_states,
    DDS_InstanceStateMask instance_states )

```

Access a collection of data samples from the **DDS_DataReader** (p. 615).

See also

FooDataReader_read (p. 625)

5.126.3.45 DDS_KeyedOctetsDataReader_take()

```

DDS_ReturnCode_t DDS_KeyedOctetsDataReader_take (
    DDS_KeyedOctetsDataReader * self,
    struct DDS_KeyedOctetsSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    DDS_SampleStateMask sample_mask,
    DDS_ViewStateMask view_mask,
    DDS_InstanceStateMask instance_mask )

```

Access a collection of data-samples from the **DDS_DataReader** (p. 615).

See also

FooDataReader_take (p. 626)

5.126.3.46 DDS_KeyedOctetsDataReader_read_w_condition()

```

DDS_ReturnCode_t DDS_KeyedOctetsDataReader_read_w_condition (
    DDS_KeyedOctetsDataReader * self,
    struct DDS_KeyedOctetsSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    DDS_ReadCondition * condition )

```

Accesses via **DDS_KeyedOctetsDataReader_read** (p. 1192) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).

See also

FooDataReader_read_w_condition (p. 630)

5.126.3.47 DDS_KeyedOctetsDataReader_take_w_condition()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataReader_take_w_condition (
    DDS_KeyedOctetsDataReader * self,
    struct DDS_KeyedOctetsSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    DDS_ReadCondition * condition )
```

Analogous to **DDS_KeyedOctetsDataReader_read_w_condition** (p. 1193) except it accesses samples via the **DDS_←_KeyedOctetsDataReader_take** (p. 1193) operation.

See also

FooDataReader_take_w_condition (p. 632)

5.126.3.48 DDS_KeyedOctetsDataReader_read_next_sample()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataReader_read_next_sample (
    DDS_KeyedOctetsDataReader * self,
    DDS_KeyedOctets * received_data,
    struct DDS_SampleInfo * sample_info )
```

Copies the next not-previously-accessed data value from the **DDS_DataReader** (p. 615).

See also

FooDataReader_read_next_sample (p. 633)

5.126.3.49 DDS_KeyedOctetsDataReader_take_next_sample()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataReader_take_next_sample (
    DDS_KeyedOctetsDataReader * self,
    DDS_KeyedOctets * received_data,
    struct DDS_SampleInfo * sample_info )
```

Copies the next not-previously-accessed data value from the **DDS_DataReader** (p. 615).

See also

FooDataReader_take_next_sample (p. 634)

5.126.3.50 DDS_KeyedOctetsDataReader_read_instance()

```

DDS_ReturnCode_t DDS_KeyedOctetsDataReader_read_instance (
    DDS_KeyedOctetsDataReader * self,
    struct DDS_KeyedOctetsSeq * received_data,
    struct 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 )

```

Access a collection of data samples from the **DDS_DataReader** (p. 615).

See also

FooDataReader_read_instance (p. 635)

5.126.3.51 DDS_KeyedOctetsDataReader_take_instance()

```

DDS_ReturnCode_t DDS_KeyedOctetsDataReader_take_instance (
    DDS_KeyedOctetsDataReader * self,
    struct DDS_KeyedOctetsSeq * received_data,
    struct 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 )

```

Access a collection of data samples from the **DDS_DataReader** (p. 615).

See also

FooDataReader_take_instance (p. 636)

5.126.3.52 DDS_KeyedOctetsDataReader_read_instance_w_condition()

```

DDS_ReturnCode_t DDS_KeyedOctetsDataReader_read_instance_w_condition (
    DDS_KeyedOctetsDataReader * self,
    struct DDS_KeyedOctetsSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    const DDS_InstanceHandle_t * a_handle,
    DDS_ReadCondition * condition )

```

Accesses via **DDS_KeyedOctetsDataReader_read_instance** (p. 1194) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).

See also

FooDataReader_read_instance_w_condition (p. 638)

5.126.3.53 DDS_KeyedOctetsDataReader_take_instance_w_condition()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataReader_take_instance_w_condition (
    DDS_KeyedOctetsDataReader * self,
    struct DDS_KeyedOctetsSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    const DDS_InstanceHandle_t * a_handle,
    DDS_ReadCondition * condition )
```

Accesses via **DDS_KeyedOctetsDataReader_take_instance** (p. 1195) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).

See also

FooDataReader_take_instance_w_condition (p. 639)

5.126.3.54 DDS_KeyedOctetsDataReader_read_next_instance()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataReader_read_next_instance (
    DDS_KeyedOctetsDataReader * self,
    struct DDS_KeyedOctetsSeq * received_data,
    struct 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 )
```

Access a collection of data samples from the **DDS_DataReader** (p. 615).

See also

FooDataReader_read_next_instance (p. 640)

5.126.3.55 DDS_KeyedOctetsDataReader_take_next_instance()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataReader_take_next_instance (
    DDS_KeyedOctetsDataReader * self,
    struct DDS_KeyedOctetsSeq * received_data,
    struct 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 )
```

Access a collection of data samples from the **DDS_DataReader** (p. 615).

See also

FooDataReader_take_next_instance (p. 642)

5.126.3.56 DDS_KeyedOctetsDataReader_read_next_instance_w_condition()

```

DDS_ReturnCode_t DDS_KeyedOctetsDataReader_read_next_instance_w_condition (
    DDS_KeyedOctetsDataReader * self,
    struct DDS_KeyedOctetsSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    const DDS_InstanceHandle_t * previous_handle,
    DDS_ReadCondition * condition )

```

Accesses via **DDS_KeyedOctetsDataReader_read_next_instance** (p. 1196) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).

See also

FooDataReader_read_next_instance_w_condition (p. 643)

5.126.3.57 DDS_KeyedOctetsDataReader_take_next_instance_w_condition()

```

DDS_ReturnCode_t DDS_KeyedOctetsDataReader_take_next_instance_w_condition (
    DDS_KeyedOctetsDataReader * self,
    struct DDS_KeyedOctetsSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq,
    DDS_Long max_samples,
    const DDS_InstanceHandle_t * previous_handle,
    DDS_ReadCondition * condition )

```

Accesses via **DDS_KeyedOctetsDataReader_take_next_instance** (p. 1196) the samples that match the criteria specified in the **DDS_ReadCondition** (p. 694).

See also

FooDataReader_take_next_instance_w_condition (p. 645)

5.126.3.58 DDS_KeyedOctetsDataReader_return_loan()

```

DDS_ReturnCode_t DDS_KeyedOctetsDataReader_return_loan (
    DDS_KeyedOctetsDataReader * self,
    struct DDS_KeyedOctetsSeq * received_data,
    struct DDS_SampleInfoSeq * info_seq )

```

Indicates to the **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615).

See also

FooDataReader_return_loan (p. 646)

5.126.3.59 DDS_KeyedOctetsDataReader_get_key_value()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataReader_get_key_value (
    DDS_KeyedOctetsDataReader * self,
    DDS_KeyedOctets * key_holder,
    const DDS_InstanceHandle_t * handle )
```

Retrieve the instance `key` that corresponds to an instance `handle`.

See also

FooDataReader_get_key_value (p. 647)

5.126.3.60 DDS_KeyedOctetsDataReader_get_key_value_w_key()

```
DDS_ReturnCode_t DDS_KeyedOctetsDataReader_get_key_value_w_key (
    DDS_KeyedOctetsDataReader * self,
    char * key,
    const DDS_InstanceHandle_t * handle )
```

<<*extension*>> (p. 1012) Retrieve the instance `key` that corresponds to an instance `handle`.

See also

FooDataReader_get_key_value (p. 647)

5.126.3.61 DDS_KeyedOctetsDataReader_lookup_instance()

```
DDS_InstanceHandle_t DDS_KeyedOctetsDataReader_lookup_instance (
    DDS_KeyedOctetsDataReader * self,
    const DDS_KeyedOctets * key_holder )
```

Retrieve the instance `handle` that corresponds to an instance `key_holder`.

See also

FooDataReader_lookup_instance (p. 648)

5.126.3.62 DDS_KeyedOctetsDataReader_lookup_instance_w_key()

```
DDS_InstanceHandle_t DDS_KeyedOctetsDataReader_lookup_instance_w_key (
    DDS_KeyedOctetsDataReader * self,
    const char * key )
```

<<**extension**>> (p. 1012) Retrieve the instance handle that corresponds to an instance key.

See also

FooDataReader_lookup_instance (p. 648)

5.126.4 Variable Documentation

5.126.4.1 key

```
char* DDS_KeyedOctets::key
```

Instance key associated with the specified value.

5.126.4.2 length

```
int DDS_KeyedOctets::length
```

Number of octets to serialize.

5.126.4.3 value

```
unsigned char* DDS_KeyedOctets::value
```

DDS_Octets (p. 1859) array value.

5.127 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. 1204) data type.*
- **#define DDS_BOOLEAN_FALSE**
*Defines "false" value of **DDS_Boolean** (p. 1204) 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.

5.127.1 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`.

5.127.2 Macro Definition Documentation

5.127.2.1 DDS_BOOLEAN_TRUE

```
#define DDS_BOOLEAN_TRUE
```

Defines "true" value of **DDS_Boolean** (p. 1204) data type.

Examples

HelloWorld.c, and **HelloWorld_publisher.c**.

5.127.2.2 DDS_BOOLEAN_FALSE

```
#define DDS_BOOLEAN_FALSE
```

Defines "false" value of **DDS_Boolean** (p. 1204) data type.

Examples

HelloWorld.c.

5.127.3 Typedef Documentation

5.127.3.1 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 character from any byte-oriented code set.

5.127.3.2 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.

5.127.3.3 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.

5.127.3.4 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.

5.127.3.5 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.

5.127.3.6 DDS_Short

```
typedef RTICdrShort DDS_Short
```

Defines a short integer data type, equivalent to IDL/CDR `short`.

A 16-bit signed short integer value.

5.127.3.7 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.

5.127.3.8 DDS_Long

```
typedef RTICdrLong DDS_Long
```

Defines a long integer data type, equivalent to IDL/CDR `long`.

A 32-bit signed long integer value.

5.127.3.9 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.

5.127.3.10 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.

5.127.3.11 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.

5.127.3.12 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.

5.127.3.13 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.

5.127.3.14 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.

5.127.3.15 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. 1201) (1) or **DDS_BOOLEAN_FALSE** (p. 1201) (0).

Examples

HelloWorld.c.

5.127.3.16 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.

5.128 Time Support

Time and duration types and defines.

Data Structures

- 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

- **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.

5.128.1 Detailed Description

Time and duration types and defines.

5.128.2 Macro Definition Documentation

5.128.2.1 DDS_TIME_ZERO

```
#define DDS_TIME_ZERO
```

The default instant in time: zero seconds and zero nanoseconds.

5.128.3 Function Documentation

5.128.3.1 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. 1201) if the given time is equal to **DDS_TIME_ZERO** (p. 1206) or **DDS_BOOLEAN_↵_FALSE** (p. 1201) otherwise.

5.128.3.2 DDS_Time_is_invalid()

```
DDS_Boolean DDS_Time_is_invalid (
    const struct DDS_Time_t * time )
```

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the given time is not valid (i.e. is negative)

5.128.3.3 DDS_Duration_is_infinite()

```
DDS_Boolean DDS_Duration_is_infinite (
    const struct DDS_Duration_t * duration )
```

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the given duration is of infinite length.

5.128.3.4 DDS_Duration_is_auto()

```
DDS_Boolean DDS_Duration_is_auto (
    const struct DDS_Duration_t * duration )
```

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the given duration has auto value.

5.128.3.5 DDS_Duration_is_zero()

```
DDS_Boolean DDS_Duration_is_zero (
    const struct DDS_Duration_t * duration )
```

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the given duration is of zero length.

5.128.4 Variable Documentation

5.128.4.1 DDS_TIME_MAX

```
const struct DDS_Time_t DDS_TIME_MAX [extern]
```

The maximum value of time.

5.128.4.2 DDS_TIME_INVALID_SEC

```
const DDS_LongLong DDS_TIME_INVALID_SEC [extern]
```

A sentinel indicating an invalid second of time.

5.128.4.3 DDS_TIME_INVALID_NSEC

```
const DDS_UnsignedLong DDS_TIME_INVALID_NSEC [extern]
```

A sentinel indicating an invalid nano-second of time.

5.128.4.4 DDS_TIME_INVALID

```
const struct DDS_Time_t DDS_TIME_INVALID [extern]
```

A sentinel indicating an invalid time.

5.128.4.5 DDS_DURATION_INFINITE_SEC

```
const DDS_Long DDS_DURATION_INFINITE_SEC [extern]
```

An infinite second period of time.

5.128.4.6 DDS_DURATION_INFINITE_NSEC

```
const DDS_UnsignedLong DDS_DURATION_INFINITE_NSEC [extern]
```

An infinite nano-second period of time.

5.128.4.7 DDS_DURATION_INFINITE

```
const struct DDS_Duration_t DDS_DURATION_INFINITE [extern]
```

An infinite period of time.

5.128.4.8 DDS_DURATION_AUTO_SEC

```
const DDS_Long DDS_DURATION_AUTO_SEC [extern]
```

An auto second period of time.

5.128.4.9 DDS_DURATION_AUTO_NSEC

```
const DDS_UnsignedLong DDS_DURATION_AUTO_NSEC [extern]
```

An auto nano-second period of time.

5.128.4.10 DDS_DURATION_AUTO

```
const struct DDS_Duration_t DDS_DURATION_AUTO [extern]
```

Duration is automatically assigned.

5.128.4.11 DDS_DURATION_ZERO_SEC

```
const DDS_Long DDS_DURATION_ZERO_SEC [extern]
```

A zero-length second period of time.

5.128.4.12 DDS_DURATION_ZERO_NSEC

```
const DDS_UnsignedLong DDS_DURATION_ZERO_NSEC [extern]
```

A zero-length nano-second period of time.

5.128.4.13 DDS_DURATION_ZERO

```
const struct DDS_Duration_t DDS_DURATION_ZERO [extern]
```

A zero-length period of time.

5.129 GUID Support

<<*extension*>> (p. 1012) GUID type and defines.

Data Structures

- 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 Connex 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.

5.129.1 Detailed Description

<<*extension*>> (p. 1012) GUID type and defines.

5.129.2 Typedef Documentation

5.129.2.1 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.

5.129.2.2 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.

5.129.2.3 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.

5.129.2.4 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. 1954). Memory and wire representation for this type is the same as the one for **DDS RTPS_GUID_t** (p. 1954).

5.129.3 Function Documentation

5.129.3.1 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

<i>self</i>	<< <i>in</i> >> (p. 1013) This GUID.
<i>other</i>	<< <i>in</i> >> (p. 1013) The other GUID to be compared with this GUID.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two GUIDs have equal values, or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.129.3.2 DDS_GUID_compare()

```
int DDS_GUID_compare (
    const struct DDS_GUID_t * self,
    const struct DDS_GUID_t * other )
```

Compares two GUIDs.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) GUID to compare.
<i>other</i>	<< <i>in</i> >> (p. 1013) 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.

5.129.3.3 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

<i>self</i>	<< <i>in</i> >> (p. 1013) This GUID. Cannot be NULL.
<i>other</i>	<< <i>in</i> >> (p. 1013) The other GUID to be copied. Cannot be NULL.

5.129.4 Variable Documentation

5.129.4.1 DDS_GUID_AUTO

```
const struct    DDS_GUID_t DDS_GUID_AUTO [extern]
```

Indicates that RTI Connexx should choose an appropriate virtual GUID.

If this special value is assigned to **DDS_DataWriterProtocolQosPolicy::virtual_guid** (p. 1650) or **DDS_DataReaderProtocolQosPolicy::virtual_guid** (p. 1597), RTI Connexx will assign the virtual GUID automatically based on the RTPS or physical GUID.

5.129.4.2 DDS_GUID_UNKNOWN

```
const struct    DDS_GUID_t DDS_GUID_UNKNOWN [extern]
```

Unknown GUID.

5.129.4.3 DDS_GUID_ZERO

```
const struct DDS_GUID_t DDS_GUID_ZERO [extern]
```

Zero GUID.

5.130 Sequence Number Support

<<*extension*>> (p. 1012) Sequence number type and defines.

Data Structures

- struct **DDS_SequenceNumber_t**
Type for sequence number representation.

Typedefs

- typedef struct **DDS_SequenceNumber_t** **DDS_SequenceNumber_t**
Type for sequence number representation.

Functions

- void **DDS_SequenceNumber_subtract** (struct **DDS_SequenceNumber_t** *answer, const struct **DDS_SequenceNumber_t** *sn1, const struct **DDS_SequenceNumber_t** *sn2)
Stores the value (sn1 - sn2) in answer
- void **DDS_SequenceNumber_add** (struct **DDS_SequenceNumber_t** *answer, const struct **DDS_SequenceNumber_t** *sn1, const struct **DDS_SequenceNumber_t** *sn2)
Stores the value (sn1 + sn2) in answer
- void **DDS_SequenceNumber_plusplus** (struct **DDS_SequenceNumber_t** *sn)
Increases the value of the input sequence number by one
- void **DDS_SequenceNumber_minusminus** (struct **DDS_SequenceNumber_t** *sn)
Decreases the value of the input sequence number by one
- int **DDS_SequenceNumber_compare** (const struct **DDS_SequenceNumber_t** *sn1, const struct **DDS_SequenceNumber_t** *sn2)
Compares two sequence numbers.

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 Connex.

5.130.1 Detailed Description

<<*extension*>> (p. 1012) Sequence number type and defines.

5.130.2 Typedef Documentation

5.130.2.1 DDS_SequenceNumber_t

```
typedef struct DDS_SequenceNumber_t DDS_SequenceNumber_t
```

Type for *sequence* number representation.

Represents a 64-bit sequence number.

5.130.3 Function Documentation

5.130.3.1 DDS_SequenceNumber_subtract()

```
void DDS_SequenceNumber_subtract (
    struct DDS_SequenceNumber_t * answer,
    const struct DDS_SequenceNumber_t * sn1,
    const struct DDS_SequenceNumber_t * sn2 )
```

Stores the value (sn1 - sn2) in answer

Parameters

<i>answer</i>	<< <i>out</i> >> (p. 1013) Result sequence number. Cannot be NULL.
<i>sn1</i>	<< <i>in</i> >> (p. 1013) First sequence number. Cannot be NULL.
<i>sn2</i>	<< <i>in</i> >> (p. 1013) Second sequence number. Cannot be NULL.

5.130.3.2 DDS_SequenceNumber_add()

```
void DDS_SequenceNumber_add (
    struct DDS_SequenceNumber_t * answer,
```

```
const struct DDS_SequenceNumber_t * sn1,
const struct DDS_SequenceNumber_t * sn2 )
```

Stores the value (sn1 + sn2) in answer

Parameters

<i>answer</i>	<< out >> (p. 1013) Result sequence number. Cannot be NULL.
<i>sn1</i>	<< in >> (p. 1013) First sequence number. Cannot be NULL.
<i>sn2</i>	<< in >> (p. 1013) Second sequence number. Cannot be NULL.

5.130.3.3 DDS_SequenceNumber_plusplus()

```
void DDS_SequenceNumber_plusplus (
    struct DDS_SequenceNumber_t * sn )
```

Increases the value of the input sequence number by one

Parameters

<i>sn</i>	<< inout >> (p. 1013) Sequence number. Cannot be NULL.

5.130.3.4 DDS_SequenceNumber_minusminus()

```
void DDS_SequenceNumber_minusminus (
    struct DDS_SequenceNumber_t * sn )
```

Decreases the value of the input sequence number by one

Parameters

<i>sn</i>	<< inout >> (p. 1013) Sequence number. Cannot be NULL.

5.130.3.5 DDS_SequenceNumber_compare()

```
int DDS_SequenceNumber_compare (
    const struct DDS_SequenceNumber_t * sn1,
    const struct DDS_SequenceNumber_t * sn2 )
```


Compares two sequence numbers.

Parameters

<i>sn1</i>	<< <i>in</i> >> (p. 1013) Sequence number to compare. Cannot be NULL.
<i>sn2</i>	<< <i>in</i> >> (p. 1013) Sequence number to compare. Cannot be NULL.

Returns

If the two sequence numbers are equal, the function returns 0. If sn1 is greater than sn2 the function returns a positive number; otherwise, it returns a negative number.

5.130.4 Variable Documentation**5.130.4.1 DDS_SEQUENCE_NUMBER_UNKNOWN**

```
const struct DDS_SequenceNumber_t DDS_SEQUENCE_NUMBER_UNKNOWN [extern]
```

Unknown sequence number.

5.130.4.2 DDS_SEQUENCE_NUMBER_ZERO

```
const struct DDS_SequenceNumber_t DDS_SEQUENCE_NUMBER_ZERO [extern]
```

Zero value for the sequence number.

5.130.4.3 DDS_SEQUENCE_NUMBER_MAX

```
const struct DDS_SequenceNumber_t DDS_SEQUENCE_NUMBER_MAX [extern]
```

Highest, most positive value for the sequence number.

5.130.4.4 DDS_AUTO_SEQUENCE_NUMBER

```
const struct DDS_SequenceNumber_t DDS_AUTO_SEQUENCE_NUMBER [extern]
```

The sequence number is internally determined by RTI Connex.

5.131 Exception Codes

<<*extension*>> (p. 1012) 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. 2071) class.

5.131.1 Detailed Description

<<*extension*>> (p. 1012) Exception codes.

These exceptions are used for error handling by the **Type Code Support** (p. 228) API.

5.131.2 Enumeration Type Documentation

5.131.2.1 DDS_ExceptionCode_t

```
enum DDS_ExceptionCode_t
```

Error codes used by the **DDS_TypeCode** (p. 2071) 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.

Enumerator

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. 2071) that was received from a remote object. The built-in publication and subscription readers provide access to information about the remote DDS_DataWriter (p. 485) and DDS_DataReader (p. 615) entities in the distributed system. Among other things, the data from these built-in readers contains the DDS_TypeCode (p. 2071) for these entities. Modifying this received DDS_TypeCode (p. 2071) is not permitted.
DDS_BAD_MEMBER_NAME_USER_EXCEPTION_CODE ↔	The specified DDS_TypeCode (p. 2071) 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. 1220)
DDS_BAD_MEMBER_ID_USER_EXCEPTION_CODE	The specified DDS_TypeCode (p. 2071) 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. 1220)

5.132 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.

5.132.1 Detailed Description

Types of return codes.

5.132.2 Standard Return Codes

Any operation with return type **DDS_ReturnCode_t** (p. 1221) may return **DDS_RETCODE_OK** (p. 1222) **DDS_RETCODE_ERROR** (p. 1222) or **DDS_RETCODE_ILLEGAL_OPERATION** (p. 1222). Any operation that takes one or more input parameters may additionally return **DDS_RETCODE_BAD_PARAMETER** (p. 1222). Any operation on an object created from any of the factories may additionally return **DDS_RETCODE_ALREADY_DELETED** (p. 1222). Any operation that is stated as optional may additionally return **DDS_RETCODE_UNSUPPORTED** (p. 1222).

Thus, the standard return codes are:

- **DDS_RETCODE_ERROR** (p. 1222)
- **DDS_RETCODE_ILLEGAL_OPERATION** (p. 1222)
- **DDS_RETCODE_ALREADY_DELETED** (p. 1222)
- **DDS_RETCODE_BAD_PARAMETER** (p. 1222)
- **DDS_RETCODE_UNSUPPORTED** (p. 1222)

Operations that may return any of the additional return codes will state so explicitly.

5.132.3 Enumeration Type Documentation

5.132.3.1 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 illegal 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.
DDS_RETCODE_OUT_OF_RESOURCES	RTI Connexant ran out of the resources needed to complete the operation.
DDS_RETCODE_NOT_ENABLED	Operation invoked on a DDS_Entity (p. 1360) 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. 1222), 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.

5.133 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. 1227)[].*

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.

5.133.1 Detailed Description

Kinds of communication status.

Entity:

DDS_Entity (p. 1360)

QoS:

All QoS Policies (p. 1238)

Listener:

DDS_Listener (p. 1816)

Each concrete **DDS_Entity** (p. 1360) 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 function on the **DDS_Entity** (p. 1360).

When these status values change, the corresponding **DDS_StatusCondition** (p. 1371) objects are activated and the proper **DDS_Listener** (p. 1816) objects are invoked to asynchronously inform the application.

An application is notified of communication status by means of the **DDS_Listener** (p. 1816) or the **DDS_WaitSet** (p. 1371) / **DDS_Condition** (p. 1370) mechanism. The two mechanisms may be combined in the application (e.g., using **DDS_WaitSet** (p. 1371) (s) / **DDS_Condition** (p. 1370) (s) to access the data and **DDS_Listener** (p. 1816) (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 **DDS_Listener** (p. 1816) mechanism is used first and then the **DDS_WaitSet** (p. 1371) 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. 1230) and **DDS_DATA_AVAILABLE_STATUS** (p. 1230).
- *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. 1230) and **DDS_DATA_AVAILABLE_STATUS** (p. 1230)) 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"

5.133.2 Changes in Status

Associated with each one of an **DDS_Entity** (p. 1360)'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"

5.133.2.1 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. 1201) each time the application accesses the plain communication status via the proper `get_<plain communication status>()` operation on the `DDS_Entity` (p. 1360).

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. 1229) will become `TRUE` each time new deadline occurs (which increases `DDS_RequestedDeadlineMissedStatus::total_count` (p. 1946)). The value changes to `FALSE` when the application accesses the status via the corresponding `DDS_DataReader_get_requested_deadline_missed_status` (p. 682) method on the proper Entity

"Changes in \p StatusChangedFlag for plain communication status"

5.133.2.2 Changes in read communication status

For the read communication status, the `StatusChangedFlag` flag is initially set to `FALSE`. The `StatusChangedFlag` becomes `TRUE` when either a data-sample arrives or else the `DDS_ViewStateKind` (p. 712), `DDS_SampleStateKind` (p. 710), or `DDS_InstanceStateKind` (p. 714) of any existing sample changes for any reason other than a call to `FooDataReader_read` (p. 625), `FooDataReader_take` (p. 626) 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. 714) of a contained instance. This can be caused by either:
 - The arrival of the notification that an instance has been disposed by:
 - * the `DDS_DataWriter` (p. 485) that owns it if `OWNERSHIP` (p. 1301) QoS kind= `DDS_EXCLUSIVE_OWNERSHIP_QOS` (p. 1302)
 - * or by any `DDS_DataWriter` (p. 485) if `OWNERSHIP` (p. 1301) QoS kind= `DDS_SHARED_OWNERSHIP_QOS` (p. 1302)
 - The loss of liveness of the `DDS_DataWriter` (p. 485) of an instance for which there is no other `DDS_DataWriter` (p. 485).
 - The arrival of the notification that an instance has been unregistered by the only `DDS_DataWriter` (p. 485) 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. 1230) `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 `DDS_DataReader` (p. 615).

- The **DDS_DATA_ON_READERS_STATUS** (p. 1230) `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 **DDS_DataReader** (p. 615) belonging to the **DDS_↔_Subscriber** (p. 572).
 - The read or take operation (or their variants) is called on any **DDS_DataReader** (p. 615) belonging to the **DDS_Subscriber** (p. 572).

"Changes in \p `StatusChangedFlag` for read communication status"

See also

DDS_Listener (p. 1816)

DDS_WaitSet (p. 1371), **DDS_Condition** (p. 1370)

5.133.3 Macro Definition Documentation

5.133.3.1 DDS_STATUS_MASK_NONE

```
#define DDS_STATUS_MASK_NONE
```

No bits are set.

Examples

HelloWorld_publisher.c, and **HelloWorld_subscriber.c**.

5.133.3.2 DDS_STATUS_MASK_ALL

```
#define DDS_STATUS_MASK_ALL
```

All bits are set.

Examples

HelloWorld_subscriber.c.

5.133.4 Typedef Documentation

5.133.4.1 DDS_StatusMask

```
typedef DDS_UnsignedLong DDS_StatusMask
```

A bit-mask (list) of concrete status types, i.e. **DDS_StatusKind** (p. 1227)[].

The bit-mask is an efficient and compact representation of a fixed-length list of **DDS_StatusKind** (p. 1227) 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);
```

5.133.5 Enumeration Type Documentation

5.133.5.1 DDS_StatusKind

```
enum DDS_StatusKind
```

Type for *status* kinds.

Each concrete **DDS_Entity** (p. 1360) is associated with a set of **Status* objects whose values represent the communication status of that **DDS_Entity** (p. 1360).

The communication statuses whose changes can be communicated to the application depend on the **DDS_Entity** (p. 1360).

Each status value can be accessed with a corresponding function on the **DDS_Entity** (p. 1360). The changes on these status values cause activation of the corresponding **DDS_StatusCondition** (p. 1371) objects and trigger invocation of the proper **DDS_Listener** (p. 1816) objects to asynchronously inform the application. Note that not all statuses will activate the **DDS_StatusCondition** (p. 1371) or have a corresponding listener callback. Refer to the documentation of the individual statuses for that information.

See also

DDS_Entity (p. 1360), **DDS_StatusCondition** (p. 1371), **DDS_Listener** (p. 1816)

Enumerator

DDS_INCONSISTENT_TOPIC_STATUS	Another topic exists with the same name but different characteristics. Entity: DDS_Topic (p. 175) Status: DDS_InconsistentTopicStatus (p. 1808) Listener: DDS_TopicListener (p. 2041)
DDS_OFFERED_DEADLINE_MISSED_STATUS	The deadline that the DDS_DataWriter (p. 485) has committed through its DDS_DeadlineQosPolicy (p. 1684) was not respected for a specific instance. Entity: DDS_DataWriter (p. 485) QoS: DEADLINE (p. 1272) Status: DDS_OfferedDeadlineMissedStatus (p. 1861) Listener: DDS_DataWriterListener (p. 1644)

Enumerator

DDS_REQUESTED_DEADLINE_MISSED_STATUS	The deadline that the DDS_DataReader (p. 615) was expecting through its DDS_DeadlineQosPolicy (p. 1684) was not respected for a specific instance. Entity: DDS_DataReader (p. 615) QoS: DEADLINE (p. 1272) Status: DDS_RequestedDeadlineMissedStatus (p. 1946) Listener: DDS_DataReaderListener (p. 1593)
DDS_OFFERED_INCOMPATIBLE_QOS_STATUS	A QosPolicy value was incompatible with what was requested. Entity: DDS_DataWriter (p. 485) Status: DDS_OfferedIncompatibleQosStatus (p. 1862) Listener: DDS_DataWriterListener (p. 1644)
DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS	A QosPolicy value was incompatible with what is offered. Entity: DDS_DataReader (p. 615) Status: DDS_RequestedIncompatibleQosStatus (p. 1947) Listener: DDS_DataReaderListener (p. 1593)

Enumerator

DDS_SAMPLE_LOST_STATUS	A sample has been lost (i.e., was never received). Entity: DDS_DataReader (p. 615) Status: DDS_SampleLostStatus (p. 1996) Listener: DDS_DataReaderListener (p. 1593)
DDS_SAMPLE_REJECTED_STATUS	A (received) sample has been rejected. Entity: DDS_DataReader (p. 615) QoS: RESOURCE_LIMITS (p. 1325) Status: DDS_SampleRejectedStatus (p. 1997) Listener: DDS_DataReaderListener (p. 1593)
DDS_DATA_ON_READERS_STATUS	New data is available. Entity: DDS_Subscriber (p. 572) Listener: DDS_SubscriberListener (p. 2009)
DDS_DATA_AVAILABLE_STATUS	One or more new data samples have been received. Entity: DDS_DataReader (p. 615) Listener: DDS_DataReaderListener (p. 1593)

Enumerator

DDS_LIVELINESS_LOST_STATUS	The liveliness that the DDS_DataWriter (p. 485) has committed to through its DDS_LivelinessQosPolicy (p. 1823) was not respected, thus DDS_DataReader (p. 615) entities will consider the DDS_DataWriter (p. 485) as no longer alive. Entity: DDS_DataWriter (p. 485) QoS: LIVELINESS (p. 1296) Status: DDS_LivelinessLostStatus (p. 1822) Listener: DDS_DataWriterListener (p. 1644)
DDS_LIVELINESS_CHANGED_STATUS	The liveliness of one or more DDS_DataWriter (p. 485) that were writing instances read through the DDS_DataReader (p. 615) has changed. Some DDS_DataWriter (p. 485) have become alive or not_alive. Entity: DDS_DataReader (p. 615) QoS: LIVELINESS (p. 1296) Status: DDS_LivelinessChangedStatus (p. 1820) Listener: DDS_DataReaderListener (p. 1593)

Enumerator

DDS_PUBLICATION_MATCHED_STATUS	The DDS_DataWriter (p. 485) has found DDS_DataReader (p. 615) that matches the DDS_Topic (p. 175) and has compatible QoS. Entity: DDS_DataWriter (p. 485) Status: DDS_PublicationMatchedStatus (p. 1914) Listener: DDS_DataWriterListener (p. 1644)
DDS_SUBSCRIPTION_MATCHED_STATUS	The DDS_DataReader (p. 615) has found DDS_DataWriter (p. 485) that matches the DDS_Topic (p. 175) and has compatible QoS. Entity: DDS_DataReader (p. 615) Status: DDS_SubscriptionMatchedStatus (p. 2022) Listener: DDS_DataReaderListener (p. 1593)
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	<<<i>extension</i>>> (p. 1012) The local DDS_DomainParticipant (p. 77) has or is about to have an invalid identity or permissions credential. Enables a DDS_DomainParticipant (p. 77) callback that is called when the local DDS_DomainParticipant (p. 77) 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: DDS_DomainParticipant (p. 77) Listener: DDS_DomainParticipantListener (p. 1723)

Enumerator

DDS_SERVICE_REQUEST_ACCEPTED_STATUS	<<extension>> (p. 1012) A DDS_DataWriter (p. 485) has been issued a DDS_ServiceRequest (p. 2001) Enables a DDS_DataWriter (p. 485) callback that is called when a DDS_ServiceRequest (p. 2001) has been accepted and dispatched to the DataWriter. Entity: DDS_DataWriter (p. 485) Listener: DDS_DataWriterListener (p. 1644)
DDS_DATA_WRITER_APPLICATION_↔ ACKNOWLEDGMENT_STATUS	<<extension>> (p. 1012) A DDS_DataWriter (p. 485) has received an application-level acknowledgment for a sample Enables a DDS_DataWriter (p. 485) callback that is called when an application-level acknowledgment from a DDS_DataReader (p. 615) is received. The callback is called for each sample that is application-level acknowledged. Changes to this status do not trigger a DDS_StatusCondition (p. 1371). Entity: DDS_DataWriter (p. 485) Listener: DDS_DataWriterListener (p. 1644)
DDS_DATA_WRITER_INSTANCE_REPLACED_↔ STATUS	<<extension>> (p. 1012) A DDS_DataWriter (p. 485) instance has been replaced Enables a DDS_DataWriter (p. 485) callback that is called when an instance in the writer queue is replaced. Entity: DDS_DataWriter (p. 485) Listener: DDS_DataWriterListener (p. 1644)

Enumerator

DDS_RELIABLE_WRITER_CACHE_CHANGED_↔ STATUS	<<extension>> (p. 1012) 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. 1951)), or the number of samples has reached a high (see DDS_RtpsReliableWriterProtocol_t::high_↔watermark (p. 1962)) or low (see DDS_RtpsReliableWriterProtocol_t::low_watermark (p. 1961)) watermark. Entity: DDS_DataWriter (p. 485) Status: DDS_ReliableWriterCacheChangedStatus (p. 1942) Listener: DDS_DataWriterListener (p. 1644)
DDS_RELIABLE_READER_ACTIVITY_CHANGED_↔ STATUS	<<extension>> (p. 1012) 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. 1960) and DDS_ReliableReaderActivityChangedStatus (p. 1940). See also DDS_RtpsReliableWriterProtocol_t (p. 1960) DDS_ReliableReaderActivityChangedStatus (p. 1940)
DDS_DATA_WRITER_CACHE_STATUS	<<extension>> (p. 1012) The status of the writer's cache. Changes to this status do not trigger a DDS_StatusCondition (p. 1371).
DDS_DATA_WRITER_PROTOCOL_STATUS	<<extension>> (p. 1012) 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 DDS_StatusCondition (p. 1371).

Enumerator

DDS_DATA_READER_CACHE_STATUS	<< <i>extension</i> >> (p. 1012) The status of the reader's cache. Changes to this status do not trigger a DDS_StatusCondition (p. 1371).
DDS_DATA_READER_PROTOCOL_STATUS	<< <i>extension</i> >> (p. 1012) 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 DDS_StatusCondition (p. 1371).
DDS_DATA_WRITER_SAMPLE_REMOVED_STATUS	<< <i>extension</i> >> (p. 1012) A DDS_DataWriter (p. 485) has removed a sample from its queue. Enables a DDS_DataWriter (p. 485) callback that is called when a sample is removed from its queue. Entity: DDS_DataWriter (p. 485) Listener: DDS_DataWriterListener (p. 1644)

5.134 Thread Settings

The properties of a thread of execution. Consult `Platform Notes` for additional platform specific details.

Data Structures

- 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. 1237).*

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. 2030) affects processor affinity for thread-related QoS policies that apply to multiple threads.*

5.134.1 Detailed Description

The properties of a thread of execution. Consult `Platform Notes` for additional platform specific details.

5.134.2 Macro Definition Documentation

5.134.2.1 DDS_THREAD_SETTINGS_KIND_MASK_DEFAULT

```
#define DDS_THREAD_SETTINGS_KIND_MASK_DEFAULT
```

The mask of default thread options.

5.134.3 Typedef Documentation

5.134.3.1 DDS_ThreadSettingsKindMask

```
typedef DDS_UnsignedLong DDS_ThreadSettingsKindMask
```

A mask of which each bit is taken from **DDS_ThreadSettingsKind** (p. 1237).

See also

DDS_ThreadSettings_t (p. 2029)

5.134.4 Enumeration Type Documentation

5.134.4.1 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. 1236)

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.

5.134.4.2 DDS_ThreadSettingsCpuRotationKind

enum **DDS_ThreadSettingsCpuRotationKind**

Determines how **DDS_ThreadSettings_t::cpu_list** (p. 2030) affects processor affinity for thread-related QoS policies that apply to multiple threads.

5.134.5 Controlling CPU Core Affinity for RTI Threads

Most thread-related QoS settings apply to a single thread (such as for the **DDS_EventQosPolicy** (p. 1793), **DDS_↔DatabaseQosPolicy** (p. 1581), and **DDS_AsynchronousPublisherQosPolicy** (p. 1539)). However, the thread settings in the **DDS_ReceiverPoolQosPolicy** (p. 1934) 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. 1934). It is ignored within other QoS policies that include **DDS_ThreadSettings_t** (p. 2029).

If **DDS_ThreadSettings_t::cpu_list** (p. 2030) is empty, the rotation is irrelevant since no affinity adjustment will occur. Suppose instead that **DDS_ThreadSettings_t::cpu_list** (p. 2030) = {0, 1} and that the middleware creates three receive threads: {A, B, C}. If **DDS_ThreadSettings_t::cpu_rotation** (p. 2030) is **DDS_THREAD_SETTINGS_CPU_↔_NO_ROTATION** (p. 1238), 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. 2030) is **DDS_THREAD_SETTINGS_CPU_RR_ROTATION** (p. 1238), each thread will be assigned in round-robin fashion to one of the processors in **DDS_ThreadSettings_t::cpu_list** (p. 2030); 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.

5.135 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.

5.135.1 Detailed Description

Quality of Service (QoS) policies.

5.136 All QoS Policies

All Quality of Service (QoS) policies.

Modules

- **ASYNCHRONOUS_PUBLISHER**
 - <<extension>> (p. 795) Specifies the asynchronous publishing settings of the **DDS_Publisher** (p. 443) instances.
- **AVAILABILITY**
 - <<extension>> (p. 795) Configures the availability of data.
- **BATCH**
 - <<extension>> (p. 795) Batch QoS policy used to enable batching in **DDS_DataWriter** (p. 485) instances.
- **DATABASE**
 - <<extension>> (p. 795) Various threads and resource limits settings used by RTI Connext to control its internal database.
- **DATA_READER_PROTOCOL**
 - <<extension>> (p. 795) Specifies the DataReader-specific protocol QoS.
- **DATA_READER_RESOURCE_LIMITS**
 - <<extension>> (p. 795) Various settings that configure how DataReaders allocate and use physical memory for internal resources.
- **DATA_REPRESENTATION**
 - <<spec>> (p. 790) A list of data representations and compression methods supported by a **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615).
- **DATA_TAG**
 - <<spec>> (p. 790) Stores (name, value) pairs that can be used to determine access permissions.
- **DATA_WRITER_PROTOCOL**
 - <<extension>> (p. 795) Along with **DDS_WireProtocolQosPolicy** (p. 2093) and **DDS_DataReaderProtocolQosPolicy** (p. 1596), this QoS policy configures the DDS on-the-network protocol (RTPS).
- **DATA_WRITER_RESOURCE_LIMITS**
 - <<extension>> (p. 795) Various settings that configure how a **DDS_DataWriter** (p. 485) allocates and uses physical memory for internal resources.
- **DATA_WRITER_TRANSFER_MODE**
 - <<extension>> (p. 795) Specifies the DataWriter transfer mode QoS.
- **DEADLINE**
 - <<spec>> (p. 790) Expresses the maximum duration (deadline) within which an instance is expected to be updated.
- **DESTINATION_ORDER**
 - <<spec>> (p. 790) Controls the criteria used to determine the logical order among changes made by **DDS_Publisher** (p. 443) entities to the same instance of data (i.e., matching **DDS_Topic** (p. 175) and key).
- **DISCOVERY**
 - <<extension>> (p. 795) Specifies the attributes required to discover participants in the domain.
- **DISCOVERY_CONFIG**
 - <<extension>> (p. 795) Specifies the discovery configuration QoS.
- **DOMAIN_PARTICIPANT_RESOURCE_LIMITS**
 - <<extension>> (p. 795) Various settings that configure how a **DDS_DomainParticipant** (p. 77) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.
- **DURABILITY**
 - <<spec>> (p. 790) This QoS policy specifies whether or not RTI Connext will store and deliver previously published data samples to new **DDS_DataReader** (p. 615) entities that join the network later.
- **DURABILITY_SERVICE**
 - <<spec>> (p. 790) Various settings to configure the external RTI Persistence Service used by RTI Connext for DataWriters with a **DDS_DurabilityQosPolicy** (p. 1759) setting of **DDS_PERSISTENT_DURABILITY_QOS** (p. 1288) or **DDS_TRANSIENT_DURABILITY_QOS** (p. 1288).
- **ENTITY_FACTORY**

<<*spec*>> (p. 790) A QoS policy for all **DDS_Entity** (p. 1360) types that can act as factories for one or more other **DDS_Entity** (p. 1360) types.

- **ENTITY_NAME**

<<*extension*>> (p. 795) Assigns a name to a **DDS_DomainParticipant** (p. 77), **DDS_Publisher** (p. 443), **DDS_Subscriber** (p. 572), **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615). Except for **DDS_Publisher** (p. 443) and **DDS_Subscriber** (p. 572), these names are visible during the discovery process and in RTI tools to help you visualize and debug your system.

- **EVENT**

<<*extension*>> (p. 795) Configures the internal thread in a DomainParticipant that handles timed events.

- **HISTORY**

<<*spec*>> (p. 790) Specifies the behavior of RTI Connex 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. 165) during discovery.

- **LATENCY_BUDGET**

<<*spec*>> (p. 790) 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. 790) Specifies how long the data written by the **DDS_DataWriter** (p. 485) is considered valid.

- **LIVELINESS**

<<*spec*>> (p. 790) Specifies and configures the mechanism that allows **DDS_DataReader** (p. 615) entities to detect when **DDS_DataWriter** (p. 485) entities become disconnected or "dead."

- **LOCATORFILTER**

<<*extension*>> (p. 795) The QoS policy used to report the configuration of a MultiChannel DataWriter as part of **DDS_PublicationBuiltinTopicData** (p. 1905).

- **LOGGING**

<<*extension*>> (p. 795) Configures the RTI Connex logging facility.

- **MONITORING**

<<*extension*>> (p. 795) Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connex telemetry data.

- **MULTICHANNEL**

<<*extension*>> (p. 795) Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.

- **OWNERSHIP**

<<*spec*>> (p. 790) Specifies whether it is allowed for multiple **DDS_DataWriter** (p. 485) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.

- **OWNERSHIP_STRENGTH**

<<*spec*>> (p. 790) Specifies the value of the strength used to arbitrate among multiple **DDS_DataWriter** (p. 485) objects that attempt to modify the same instance of a data type (identified by **DDS_Topic** (p. 175) + key).

- **PARTITION**

<<*spec*>> (p. 790) Set of strings that introduces logical partitions in **DDS_DomainParticipant** (p. 77), **DDS_Publisher** (p. 443), or **DDS_Subscriber** (p. 572) entities.

- **PRESENTATION**

<<*spec*>> (p. 790) Specifies how the samples representing changes to data instances are presented to a subscribing application.

- **PROFILE**

<<*extension*>> (p. 795) Configures the way that XML documents containing QoS profiles are loaded by RTI Connex.

- **PROPERTY**

<<*extension*>> (p. 795) Stores (name, value) pairs that can be used to configure certain parameters of RTI Connex 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. 795) 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. 790) Controls how a DataReader manages the lifecycle of the data that it has received.

- **RECEIVER_POOL**

<<extension>> (p. 795) Configures threads used by RTI Connext to receive and process data from transports (for example, UDP sockets).

- **RELIABILITY**

<<spec>> (p. 790) Indicates the level of reliability offered/requested by RTI Connext.

- **RESOURCE_LIMITS**

<<spec>> (p. 790) 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. 795) Service QoS policy is used to indicate what kind of service or tool is associated with the DDS entity.

- **SYSTEM_RESOURCE_LIMITS**

<<extension>> (p. 795) Configures DomainParticipant-independent resources used by RTI Connext.

- **TIME_BASED_FILTER**

*<<spec>> (p. 790) Filter that allows a **DDS_DataReader** (p. 615) 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. 165) during discovery.*

- **TOPIC_QUERY_DISPATCH**

*<<extension>> (p. 795) Configures the ability of a **DDS_DataWriter** (p. 485) to publish historical samples*

- **TRANSPORT_BUILTIN**

<<extension>> (p. 795) Specifies which built-in transports are used.

- **TRANSPORT_MULTICAST**

*<<extension>> (p. 795) Specifies the multicast address on which a **DDS_DataReader** (p. 615) wants to receive its data. It can also specify a port number, as well as a subset of the available (at the **DDS_DomainParticipant** (p. 77) level) transports with which to receive the multicast data.*

- **TRANSPORT_MULTICAST_MAPPING**

<<extension>> (p. 795) 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. 790) This QoS policy allows the application to take advantage of transport that are capable of sending messages with different priorities.

- **TRANSPORT_SELECTION**

*<<extension>> (p. 795) Specifies the physical transports that a **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615) may use to send or receive data.*

- **TRANSPORT_UNICAST**

<<extension>> (p. 795) Specifies a subset of transports and a port number that can be used by an Entity to receive data.

- **TYPE_CONSISTENCY_ENFORCEMENT**

<<spec>> (p. 790) 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. 965) <<extension>> (p. 795) Allows you to attach application-specific values to a **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615), 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. 165) during discovery.*

- **WRITER_DATA_LIFECYCLE**

*<<spec>> (p. 790) Controls how a **DataWriter** handles the lifecycle of the instances (keys) that it is registered to manage.*

- **WIRE_PROTOCOL**

*<<extension>> (p. 795) Specifies the wire protocol related attributes for the **DDS_DomainParticipant** (p. 77).*

- **Extended QoS Support**

<<extension>> (p. 795) Types and defines used in extended QoS policies.

Data Structures

- 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. 1246).*

- struct **DDS_QoSPolicyCountSeq**

*Declares IDL sequence < **DDS_QoSPolicyCount** (p. 1924) >*

Macros

- #define **DDS_QoSPrintFormat_INITIALIZER**

*Static initializer for **DDS_QoSPrintFormat** (p. 1926).*

- #define **DDS_QOS_POLICY_COUNT**

*Number of QoS policies in **DDS_QoSPolicyId_t** (p. 1246).*

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 QoS Policies.

5.136.1 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 **DDS_Entity** (p. 1360) objects in the system such as **DDS_Topic** (p. 175), **DDS_DataWriter** (p. 485), **DDS_DataReader** (p. 615), **DDS_Publisher** (p. 443), **DDS_Subscriber** (p. 572), and **DDS_DomainParticipant** (p. 77).

5.136.2 Specifying QoS on entities

QoS Policies can be set programmatically when an **DDS_Entity** (p. 1360) is created, or modified with the **DDS_Entity** (p. 1360)'s **set_qos (abstract)** (p. 1361) function.

QoS Policies 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. 971).

To customize a **DDS_Entity** (p. 1360)'s QoS before creating the entity, the correct pattern is:

- First, initialize a QoS object with the appropriate `INITIALIZER` constant.
- 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. 1361) operation, or when the **DDS_Entity** (p. 1360) 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 **DDS_Entity** (p. 1360) creation time or else prior to calling the **DDS_Entity_enable** (p. 1363) operation on the entity.

Each **DDS_Entity** (p. 1360) can be configured with a list of `QosPolicy` objects. However, not all QoS Policies are supported by each **DDS_Entity** (p. 1360). For instance, a **DDS_DomainParticipant** (p. 77) supports a different set of QoS Policies than a **DDS_Topic** (p. 175) or a **DDS_Publisher** (p. 443).

Additional properties that are not exposed through the formal QoS policies can also be set for an **DDS_Entity** (p. 1360) via the **DDS_PropertyQosPolicy** (p. 1901). These properties are described in the `Property Reference Guide`.

5.136.3 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 **DDS_Subscriber** (p. 572) requests to receive data reliably while the corresponding **DDS_Publisher** (p. 443) 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 specifies 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 **DDS_Entity** (p. 1360) objects and will record this fact by means of the **DDS_OFFERED_INCOMPATIBLE_QOS_STATUS** (p. 1229) on the publisher end and **DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 1229) on the subscriber end. The application can detect this fact by means of a **DDS_Listener** (p. 1816) or a **DDS_Condition** (p. 1370).

The following **properties** are defined on a `QosPolicy`.

- **RxO (p. 1245) 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. 1245) property:

- **RxO (p. 1245) = 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. 1245) = 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. 1245) = 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. 1245) property**

Determines whether a `QosPolicy` can be changed.

- **NO (p. 1245)** -- policy can only be specified at **DDS_Entity** (p. 1360) creation time.
- **UNTIL ENABLE (p. 1245)** -- policy can only be changed before the **DDS_Entity** (p. 1360) is enabled.
- **YES (p. 1245)** -- policy can be changed at any time.

5.136.4 Macro Definition Documentation

5.136.4.1 DDS_QosPrintFormat_INITIALIZER

```
#define DDS_QosPrintFormat_INITIALIZER
```

Static initializer for **DDS_QosPrintFormat** (p. 1926).

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. 1926)

5.136.4.2 DDS_QOS_POLICY_COUNT

```
#define DDS_QOS_POLICY_COUNT
```

Number of QoS policies in **DDS_QosPolicyId_t** (p. 1246).

5.136.5 Enumeration Type Documentation

5.136.5.1 DDS_QosPolicyId_t

```
enum DDS_QosPolicyId_t
```

Type to identify QoS Policies.

Enumerator

DDS_INVALID_QOS_POLICY_ID	Identifier for an invalid QoS policy.
DDS_USERDATA_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_UserDataQosPolicy (p. 2086)
DDS_DURABILITY_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_DurabilityQosPolicy (p. 1759)
DDS_PRESENTATION_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_PresentationQosPolicy (p. 1890)
DDS_DEADLINE_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_DeadlineQosPolicy (p. 1684)
DDS_LATENCYBUDGET_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_LatencyBudgetQosPolicy (p. 1813)
DDS_OWNERSHIP_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_OwnershipQosPolicy (p. 1863)

Enumerator

DDS_OWNERSHIPSTRENGTH_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_OwnershipStrengthQosPolicy (p. 1868)
DDS_LIVELINESS_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_LivelinessQosPolicy (p. 1823)
DDS_TIMEBASEDFILTER_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_TimeBasedFilterQosPolicy (p. 2032)
DDS_PARTITION_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_PartitionQosPolicy (p. 1881)
DDS_RELIABILITY_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_ReliabilityQosPolicy (p. 1936)
DDS_DESTINATIONORDER_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_DestinationOrderQosPolicy (p. 1686)
DDS_HISTORY_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_HistoryQosPolicy (p. 1805)
DDS_RESOURCELIMITS_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_ResourceLimitsQosPolicy (p. 1949)
DDS_ENTITYFACTORY_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_EntityFactoryQosPolicy (p. 1788)
DDS_WRITERDATA_LIFECYCLE_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_WriterDataLifecycleQosPolicy (p. 2106)
DDS_READERDATA_LIFECYCLE_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_ReaderDataLifecycleQosPolicy (p. 1930)
DDS_TOPICDATA_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_TopicDataQosPolicy (p. 2040)
DDS_GROUPDATA_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_GroupDataQosPolicy (p. 1803)
DDS_TRANSPORTPRIORITY_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_TransportPriorityQosPolicy (p. 2062)
DDS_LIFESPAN_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_LifespanQosPolicy (p. 1815)
DDS_DURABILITYSERVICE_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_DurabilityServiceQosPolicy (p. 1764)
DDS_DATA_REPRESENTATION_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_DataRepresentationQosPolicy (p. 1638)
DDS_TYPE_CONSISTENCY_ENFORCEMENT_QOS_POLICY_ID	<< <i>spec</i> >> (p. 790) Identifier for DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)
DDS_DATATAG_QOS_POLICY_ID	Identifier for DDS_DataTagQosPolicy (p. 1264).
DDS_WIREPROTOCOL_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_WireProtocolQosPolicy (p. 2093)
DDS_DISCOVERY_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_DiscoveryQosPolicy (p. 1715)
DDS_DATAREADERRESOURCELIMITS_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_DataReaderResourceLimitsQosPolicy (p. 1619)
DDS_DATAWRITERRESOURCELIMITS_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_DataWriterResourceLimitsQosPolicy (p. 1673)

Enumerator

DDS_DATAREADERPROTOCOL_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_DataReaderProtocolQosPolicy (p. 1596)
DDS_DATAWRITERPROTOCOL_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_DataWriterProtocolQosPolicy (p. 1649)
DDS_DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)
DDS_EVENT_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_EventQosPolicy (p. 1793)
DDS_DATABASE_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_DatabaseQosPolicy (p. 1581)
DDS_RECEIVERPOOL_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_ReceiverPoolQosPolicy (p. 1934)
DDS_DISCOVERYCONFIG_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_DiscoveryConfigQosPolicy (p. 1689)
DDS_SYSTEMRESOURCELIMITS_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_SystemResourceLimitsQosPolicy (p. 2024)
DDS_TRANSPORTSELECTION_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_TransportSelectionQosPolicy (p. 2064)
DDS_TRANSPORTUNICAST_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_TransportUnicastQosPolicy (p. 2065)
DDS_TRANSPORTMULTICAST_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_TransportMulticastQosPolicy (p. 2058)
DDS_TRANSPORTBUILTIN_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_TransportBuiltinQosPolicy (p. 2051)
DDS_TYPESUPPORT_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_TypeSupportQosPolicy (p. 2081)
DDS_PROPERTY_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_PropertyQosPolicy (p. 1901)
DDS_PUBLISHMODE_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_PublishModeQosPolicy (p. 1920)
DDS_ASYNCHRONOUSPUBLISHER_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_AsynchronousPublisherQosPolicy (p. 1539)
DDS_ENTITYNAME_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_EntityNameQosPolicy (p. 1790)
DDS_BATCH_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_BatchQosPolicy (p. 1552)
DDS_PROFILE_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_ProfileQosPolicy (p. 1898)
DDS_LOCATORFILTER_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_LocatorFilterQosPolicy (p. 1829)
DDS_MULTICHANNEL_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_MultiChannelQosPolicy (p. 1856)
DDS_AVAILABILITY_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_AvailabilityQosPolicy (p. 1547)
DDS_TRANSPORTMULTICASTMAPPING_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_TransportMulticastMappingQosPolicy (p. 2056)

Enumerator

DDS_LOGGING_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_LoggingQosPolicy (p. 1831)
DDS_TOPICQUERYDISPATCH_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_TopicQueryDispatchQosPolicy (p. 2047)
DDS_DATAWRITERTRANSFERMODE_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_DataWriterTransferModeQosPolicy (p. 1683)
DDS_MONITORING_QOS_POLICY_ID	<< <i>extension</i> >> (p. 795) Identifier for DDS_MonitoringQosPolicy (p. 1853)

5.137 QoS by Category

Quality of Service (QoS) organized into categories.

Modules

- **Basic QoS**

QoS that are considered basic and fundamental to RTI Connex.

- **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. 871), 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. 949) or **immutable** (p. 914), or if they **apply per-instance** (p. 956).*

- **Deprecated QoS**

Deprecated QoS.

5.137.1 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. 871), then looking at any QoS in that category will be helpful. The different functional categories are listed on **Functional Categories** (p. 833).

Some categories are not necessarily related to achieving a specific functionality, but are characteristics of the QoS – for example if they are **mutable** (p. 949) or **immutable** (p. 914), or if they **apply per-instance** (p. 956). These categories are listed under **QoS Mutability** (p. 914) and **Other QoS Attributes** (p. 956).

5.138 ASYNCHRONOUS_PUBLISHER

<<*extension*>> (p. 795) Specifies the asynchronous publishing settings of the **DDS_Publisher** (p. 443) instances.

Data Structures

- struct **DDS_AsynchronousPublisherQosPolicy**

<<*extension*>> (p. 795) *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. 1539).*

5.138.1 Detailed Description

<<*extension*>> (p. 795) Specifies the asynchronous publishing settings of the **DDS_Publisher** (p. 443) instances.

5.138.2 Variable Documentation

5.138.2.1 DDS_ASYNCHRONOUSPUBLISHER_QOS_POLICY_NAME

```
const char* const DDS_ASYNCHRONOUSPUBLISHER_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_AsynchronousPublisherQosPolicy** (p. 1539).

5.139 AVAILABILITY

<<*extension*>> (p. 795) Configures the availability of data.

Data Structures

- 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. 1784).*

- struct **DDS_AvailabilityQosPolicy**

<<*extension*>> (p. 795) *Configures the availability of data.*

Variables

- `const char *const DDS_AVAILABILITY_QOS_POLICY_NAME`
*Stringified human-readable name for **DDS_AvailabilityQosPolicy** (p. 1547).*

5.139.1 Detailed Description

<<**extension**>> (p. 795) Configures the availability of data.

5.139.2 Variable Documentation

5.139.2.1 DDS_AVAILABILITY_QOS_POLICY_NAME

```
const char* const DDS_AVAILABILITY_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_AvailabilityQosPolicy** (p. 1547).

5.140 BATCH

<<**extension**>> (p. 795) Batch QoS policy used to enable batching in **DDS_DataWriter** (p. 485) instances.

Data Structures

- `struct DDS_BatchQosPolicy`
<<**extension**>> (p. 795) *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. 1552).*

5.140.1 Detailed Description

<<**extension**>> (p. 795) Batch QoS policy used to enable batching in **DDS_DataWriter** (p. 485) instances.

5.140.2 Variable Documentation

5.140.2.1 DDS_BATCH_QOS_POLICY_NAME

```
const char* const DDS_BATCH_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_BatchQosPolicy** (p. 1552).

5.141 DATABASE

<<**extension**>> (p. 795) Various threads and resource limits settings used by RTI Connex to control its internal database.

Data Structures

- struct **DDS_DatabaseQosPolicy**
<<**extension**>> (p. 795) Various threads and resource limits settings used by RTI Connex to control its internal database.

Variables

- const char *const **DDS_DATABASE_QOS_POLICY_NAME**
Stringified human-readable name for **DDS_DatabaseQosPolicy** (p. 1581).

5.141.1 Detailed Description

<<**extension**>> (p. 795) Various threads and resource limits settings used by RTI Connex to control its internal database.

5.141.2 Variable Documentation

5.141.2.1 DDS_DATABASE_QOS_POLICY_NAME

```
const char* const DDS_DATABASE_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_DatabaseQosPolicy** (p. 1581).

5.142 DATA_READER_PROTOCOL

<<*extension*>> (p. 795) Specifies the DataReader-specific protocol QoS.

Data Structures

- struct **DDS_DataReaderProtocolQosPolicy**

<<*extension*>> (p. 795) Along with **DDS_WireProtocolQosPolicy** (p. 2093) and **DDS_DataWriterProtocolQosPolicy** (p. 1649), 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. 1596).

5.142.1 Detailed Description

<<*extension*>> (p. 795) Specifies the DataReader-specific protocol QoS.

5.142.2 Variable Documentation

5.142.2.1 DDS_DATAREADERPROTOCOL_QOS_POLICY_NAME

```
const char* const DDS_DATAREADERPROTOCOL_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_DataReaderProtocolQosPolicy** (p. 1596).

5.143 DATA_READER_RESOURCE_LIMITS

<<*extension*>> (p. 795) Various settings that configure how DataReaders allocate and use physical memory for internal resources.

Data Structures

- struct **DDS_DataReaderResourceLimitsInstanceReplacementSettings**

Instance replacement kind applied to each instance state.

- struct **DDS_DataReaderResourceLimitsQosPolicy**

<<*extension*>> (p. 795) Various settings that configure how a **DDS_DataReader** (p. 615) 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. 1951)) are reached.

Variables

- const char *const **DDS_DATAREADERRESOURCELIMITS_QOS_POLICY_NAME**
Stringified human-readable name for **DDS_DataReaderResourceLimitsQosPolicy** (p. 1619).
- const **DDS_Long** **DDS_AUTO_MAX_TOTAL_INSTANCES**
<<extension>> (p. 795) This value is used to make **DDS_DataReaderResourceLimitsQosPolicy::max_total_instances** (p. 1629) equal to **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1951).

5.143.1 Detailed Description

<<extension>> (p. 795) Various settings that configure how DataReaders allocate and use physical memory for internal resources.

5.143.2 Enumeration Type Documentation

5.143.2.1 DDS_DataReaderInstanceRemovalKind

```
enum DDS_DataReaderInstanceRemovalKind
```

Sets the kinds of instances that can be replaced when instance resource limits (**DDS_ResourceLimitsQosPolicy::max_instances** (p. 1951)) are reached.

See also

DDS_DataReaderResourceLimitsQosPolicy::instance_replacement (p. 1635)

Enumerator

DDS_NO_INSTANCE_REMOVAL	No instance can be removed. If an instance resource is required because DDS_ResourceLimitsQosPolicy::max_instances (p. 1951) 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. 618).
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. 1815) or sample purging due to the DDS_ReaderDataLifecycleQosPolicy (p. 1930), 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. 1939): • DDS_RELIABLE_RELIABILITY_QOS (p. 1323) (depends on the DDS_ReliabilityQosPolicy↔ AcknowledgmentModeKind (p. 1323)): – DDS_PROTOCOL↔ ACKNOWLEDGMENT_MODE (p. 1323) or DDS_APPLICATION_AUTO↔ ACKNOWLEDGMENT_MODE (p. 1323): The sample is considered processed when it has been read or taken by the application and <code>return_loan</code> has been called. – DDS_APPLICATION_EXPLICIT↔ ACKNOWLEDGMENT_MODE (p. 1324): The sample is considered processed when the subscribing application has explicitly acknowledged the DDS sample, the <code>AppAckConf</code> has been received, and the application has called <code>return_loan</code>. • DDS_BEST_EFFORT_RELIABILITY_QOS (p. 1323): All samples are considered processed when they have been read or taken by the application and <code>return_loan</code> 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.

5.143.3 Variable Documentation

5.143.3.1 DDS_DATAREADERRESOURCELIMITS_QOS_POLICY_NAME

```
const char* const DDS_DATAREADERRESOURCELIMITS_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_DataReaderResourceLimitsQosPolicy** (p. 1619).

5.143.3.2 DDS_AUTO_MAX_TOTAL_INSTANCES

```
const DDS_Long DDS_AUTO_MAX_TOTAL_INSTANCES [extern]
```

<<*extension*>> (p. 795) This value is used to make **DDS_DataReaderResourceLimitsQosPolicy::max_total_instances** (p. 1629) equal to **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1951).

5.144 DATA REPRESENTATION

<<*spec*>> (p. 790) A list of data representations and compression methods supported by a **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615).

Modules

- **Compression Settings**

<<*extension*>> (p. 795) *Compression settings that can be applied to data that is sent and received by DataWriters and DataReaders.*

Data Structures

- struct **DDS_DataRepresentationIdSeq**

Declares IDL sequence < DDS_DataRepresentationId_t (p. 1257) >

- struct **DDS_DataRepresentationQosPolicy**

<<*spec*>> (p. 790) *This QoS policy contains a list of representation identifiers and compression settings used by DDS_DataWriter (p. 485) and DDS_DataReader (p. 615) 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. 1638).*

5.144.1 Detailed Description

<<*spec*>> (p. 790) A list of data representations and compression methods supported by a **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615).

5.144.2 Typedef Documentation

5.144.2.1 DDS_DataRepresentationId_t

```
typedef DDS_Short DDS_DataRepresentationId_t
```

A two-byte signed integer that identifies a data representation.

5.144.3 Variable Documentation

5.144.3.1 DDS_XCDR_DATA_REPRESENTATION

```
const DDS_DataRepresentationId_t DDS_XCDR_DATA_REPRESENTATION [extern]
```

Corresponds to the Extended Common Data Representation encoding version 1.

5.144.3.2 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.

5.144.3.3 DDS_XCDR2_DATA_REPRESENTATION

```
const DDS_DataRepresentationId_t DDS_XCDR2_DATA_REPRESENTATION [extern]
```

Corresponds to the Extended Common Data Representation encoding version 2.

5.144.3.4 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 Connexx translates this field to **DDS_XCDR_DATA_REPRESENTATION** (p. 1257). Otherwise, it translates this field to **DDS_XCDR2_DATA_REPRESENTATION** (p. 1258).

For the **FlatData language binding** (p.209), RTI Connexx translates this field to **DDS_XCDR2_DATA_REPRESENTATION** (p. 1258).

Examples

HelloWorldPlugin.c.

5.144.3.5 DDS_DATA_REPRESENTATION_QOS_POLICY_NAME

```
const char* const DDS_DATA_REPRESENTATION_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_DataRepresentationQosPolicy** (p. 1638).

5.145 Compression Settings

<<*extension*>> (p. 795) Compression settings that can be applied to data that is sent and received by DataWriters and DataReaders.

Data Structures

- struct **DDS_CompressionSettings_t**
 <<*extension*>> (p. 795) Settings related to compressing user data.

Macros

- #define **DDS_COMPRESSION_ID_MASK_NONE** ((DDS_CompressionIdMask) (0))
DDS_CompressionIdMask (p. 1262) used to disable user-data compression for an endpoint.
- #define **DDS_COMPRESSION_ID_MASK_ALL**
DDS_CompressionIdMask (p. 1262) mask with all the supported 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. 1262) mask with the default value for publication.
- #define **DDS_COMPRESSION_ID_MASK_SUBSCRIPTION_DEFAULT**
DDS_CompressionIdMask (p. 1262) 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.

5.145.1 Detailed Description

<<*extension*>> (p. 795) Compression settings that can be applied to data that is sent and received by DataWriters and DataReaders.

5.145.2 Macro Definition Documentation

5.145.2.1 DDS_COMPRESSION_ID_MASK_NONE

```
#define DDS_COMPRESSION_ID_MASK_NONE (( DDS_CompressionIdMask) (0))
```

DDS_CompressionIdMask (p. 1262) used to disable user-data compression for an endpoint.

5.145.2.2 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. 1262) mask with all the supported built-in compression algorithms enabled.

5.145.2.3 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. 1569)

5.145.2.4 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. 1569)

5.145.2.5 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. 1569)

5.145.2.6 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. 1569)

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.

5.145.2.7 DDS_COMPRESSION_ID_MASK_PUBLICATION_DEFAULT

```
#define DDS_COMPRESSION_ID_MASK_PUBLICATION_DEFAULT DDS_COMPRESSION_ID_MASK_NONE
```

DDS_CompressionIdMask (p. 1262) mask with the default value for publication.

See also

DDS_CompressionSettings_t::compression_ids (p. 1568)

DDS_COMPRESSION_ID_MASK_SUBSCRIPTION_DEFAULT (p. 1261)

5.145.2.8 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. 1262) mask with the default value for subscription.

See also

DDS_CompressionSettings_t::compression_ids (p. 1568)

DDS_COMPRESSION_ID_MASK_PUBLICATION_DEFAULT (p. 1261)

5.145.3 Typedef Documentation

5.145.3.1 DDS_CompressionId_t

```
typedef DDS_UnsignedLong DDS_CompressionId_t
```

A four-byte signed integer that identifies a single compression algorithm.

5.145.3.2 DDS_CompressionIdMask

```
typedef DDS_UnsignedLong DDS_CompressionIdMask
```

A four-byte signed integer that identifies a mask of the compression IDs that the algorithms enabled.

5.145.4 Variable Documentation

5.145.4.1 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.

5.145.4.2 DDS_COMPRESSION_ID_BZIP2

```
const DDS_CompressionId_t DDS_COMPRESSION_ID_BZIP2 [extern]
```

Corresponds to the ID of the compression algorithm BZIP2.

5.145.4.3 DDS_COMPRESSION_ID_LZ4

```
const DDS_CompressionId_t DDS_COMPRESSION_ID_LZ4 [extern]
```

Corresponds to the ID of the compression algorithm LZ4.

5.146 DATA_TAG

<<*spec*>> (p. 790) Stores (name, value) pairs that can be used to determine access permissions.

Data Structures

- struct **DDS_Tag**
Tags are name/value pair objects.
- struct **DDS_TagSeq**
Declares IDL sequence < DDS_Tag (p. 2026) >
- struct **DDS_DataTags**
Definition of DDS_DataTagQosPolicy (p. 1264).

Typedefs

- typedef struct **DDS_DataTags DDS_DataTagQosPolicy**
<<spec>> (p. 790) Stores (name, value) pairs that can be used to determine access permissions.

Functions

- struct **DDS_Tag * DDS_DataTagQosPolicyHelper_lookup_tag** (const **DDS_DataTagQosPolicy** *policy, const char *name)
Searches by tag name for a tag in the input policy.
- **DDS_ReturnCode_t DDS_DataTagQosPolicyHelper_assert_tag** (**DDS_DataTagQosPolicy** *policy, const char *name, const char *value)
Asserts the tag identified by name in the input policy.
- **DDS_ReturnCode_t DDS_DataTagQosPolicyHelper_add_tag** (**DDS_DataTagQosPolicy** *policy, const char *name, const char *value)
Adds a new tag to the input policy.
- **DDS_ReturnCode_t DDS_DataTagQosPolicyHelper_remove_tag** (**DDS_DataTagQosPolicy** *policy, const char *name)
Removes a tag from the input policy.
- **DDS_Long DDS_DataTagQosPolicyHelper_get_number_of_tags** (const **DDS_DataTagQosPolicy** *policy)
Gets the number of data tags in the input policy.

Variables

- `const char *const DDS_DATATAG_QOS_POLICY_NAME`
*Stringified human-readable name for **DDS_DataTagQosPolicy** (p. 1264).*

5.146.1 Detailed Description

<<*spec*>> (p. 790) Stores (name, value) pairs that can be used to determine access permissions.

The **DDS_DataTagQosPolicy** (p. 1264) can be used to associate a set of tags in the form of (name, value) pairs with a **DDS_DataReader** (p. 615) or **DDS_DataWriter** (p. 485). This is similar to the **DDS_PropertyQosPolicy** (p. 1901), 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.

5.146.2 Typedef Documentation

5.146.2.1 DDS_DataTagQosPolicy

```
typedef struct DDS_DataTags DDS_DataTagQosPolicy
```

<<*spec*>> (p. 790) Stores (name, value) pairs that can be used to determine access permissions.

Entity:

DDS_DataReader (p. 615) **DDS_DataWriter** (p. 485)

Properties:

RxO (p. 1245) = N/A;
Changeable (p. 1245) = **NO** (p. 1245)

5.146.3 Usage

The **DATA_TAG** QoS policy can be used to associate a set of tags in the form of (name, value) pairs with a **DDS_DataReader** (p. 615) or **DDS_DataWriter** (p. 485). This is similar to the **DDS_PropertyQosPolicy** (p. 1901), 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 **DDS_DataWriter_get_matched_subscription_data** (p. 540), that receive **DDS_PublicationBuiltinTopicData** (p. 1905) or **DDS_SubscriptionBuiltinTopicData** (p. 2012) as a parameter.
- Connex 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. 1263) page.

5.146.4 Function Documentation

5.146.4.1 DDS_DataTagQosPolicyHelper_lookup_tag()

```
struct DDS_Tag * DDS_DataTagQosPolicyHelper_lookup_tag (
    const DDS_DataTagQosPolicy * policy,
    const char * name )
```

Searches by tag name for a tag in the input policy.

Precondition

policy, name and value cannot be NULL.

Parameters

<i>policy</i>	<< <i>in</i> >> (p. 1013) Input policy.
<i>name</i>	<< <i>in</i> >> (p. 1013) Tag name.

Returns

The function returns the first tag with the given name. If such a tag does not exist, the function returns NULL.

5.146.4.2 DDS_DataTagQosPolicyHelper_assert_tag()

```
DDS_ReturnCode_t DDS_DataTagQosPolicyHelper_assert_tag (
    DDS_DataTagQosPolicy * policy,
    const char * name,
    const char * value )
```

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

<i>policy</i>	<< <i>in</i> >> (p. 1013) Input policy.
<i>name</i>	<< <i>in</i> >> (p. 1013) Tag name.
<i>value</i>	<< <i>in</i> >> (p. 1013) Tag value.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

5.146.4.3 DDS_DataTagQosPolicyHelper_add_tag()

```
DDS_ReturnCode_t DDS_DataTagQosPolicyHelper_add_tag (
    DDS_DataTagQosPolicy * policy,
    const char * name,
    const char * value )
```

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. 1222).

Precondition

policy, name and value cannot be NULL.

The tag is not in the policy.

Parameters

<i>policy</i>	<< <i>in</i> >> (p. 1013) Input policy.
<i>name</i>	<< <i>in</i> >> (p. 1013) Tag name.
<i>value</i>	<< <i>in</i> >> (p. 1013) Tag value.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222), **DDS↵_RETCODE_OUT_OF_RESOURCES** (p. 1222)

5.146.4.4 DDS_DataTagQosPolicyHelper_remove_tag()

```
DDS_ReturnCode_t DDS_DataTagQosPolicyHelper_remove_tag (
    DDS_DataTagQosPolicy * policy,
    const char * name )
```

Removes a tag from the input policy.

If the tag does not exist, the function fails with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

Precondition

policy and name cannot be NULL.

The tag is in the policy.

Parameters

<i>policy</i>	<< <i>in</i> >> (p. 1013) Input policy.
<i>name</i>	<< <i>in</i> >> (p. 1013) Tag name.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222)

5.146.4.5 DDS_DataTagQosPolicyHelper_get_number_of_tags()

```
DDS_Long DDS_DataTagQosPolicyHelper_get_number_of_tags (
    const DDS_DataTagQosPolicy * policy )
```

Gets the number of data tags in the input policy.

Precondition

policy cannot be NULL.

Parameters

<i>policy</i>	<< <i>in</i> >> (p. 1013) Input policy.

Returns

Number of data tags.

5.146.5 Variable Documentation

5.146.5.1 DDS_DATATAG_QOS_POLICY_NAME

```
const char* const DDS_DATATAG_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_DataTagQosPolicy** (p. 1264).

5.147 DATA_WRITER_PROTOCOL

<<*extension*>> (p. 795) Along with **DDS_WireProtocolQosPolicy** (p. 2093) and **DDS_DataReaderProtocolQosPolicy** (p. 1596), this QoS policy configures the DDS on-the-network protocol (RTPS).

Data Structures

- struct **DDS_DataWriterProtocolQosPolicy**
 <<*extension*>> (p. 795) Protocol that applies only to **DDS_DataWriter** (p. 485) instances.

Variables

- const char *const **DDS_DATAWRITERPROTOCOL_QOS_POLICY_NAME**
 Stringified human-readable name for **DDS_DataWriterProtocolQosPolicy** (p. 1649).

5.147.1 Detailed Description

<<*extension*>> (p. 795) Along with **DDS_WireProtocolQosPolicy** (p. 2093) and **DDS_DataReaderProtocolQosPolicy** (p. 1596), this QoS policy configures the DDS on-the-network protocol (RTPS).

5.147.2 Variable Documentation

5.147.2.1 DDS_DATAWRITERPROTOCOL_QOS_POLICY_NAME

```
const char* const DDS_DATAWRITERPROTOCOL_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_DataWriterProtocolQosPolicy** (p. 1649).

5.148 DATA_WRITER_RESOURCE_LIMITS

<<*extension*>> (p. 795) Various settings that configure how a **DDS_DataWriter** (p. 485) allocates and uses physical memory for internal resources.

Data Structures

- struct **DDS_DataWriterResourceLimitsQosPolicy**
 <<*extension*>> (p. 795) Various settings that configure how a **DDS_DataWriter** (p. 485) 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. 1673).

5.148.1 Detailed Description

<<*extension*>> (p. 795) Various settings that configure how a **DDS_DataWriter** (p. 485) allocates and uses physical memory for internal resources.

5.148.2 Enumeration Type Documentation

5.148.2.1 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. 1951) is reached, a **DDS_DataWriter** (p. 485) 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. 1677).

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 **DDS_DataWriter** (p. 485) will replace the oldest instance satisfying that kind. For example, when the kind is **DDS_UNREGISTERED_INSTANCE_REPLACEMENT** (p. 1270), a **DDS_DataWriter** (p. 485) 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 **DDS_DataWriter** (p. 485) will replace the old instance with the new instance, and invoke, if available, the **DDS_DataWriterListener::on_instance_replaced** (p. 1647) to notify the user about an instance being replaced.

A **DDS_DataWriter** (p. 485) checks for replaceable instances in the following order, stopping once a replaceable instance is found:

- If **DDS_DataWriterResourceLimitsQosPolicy::replace_empty_instances** (p. 1678) is **DDS_BOOLEAN_TRUE** (p. 1201), a **DDS_DataWriter** (p. 485) first tries replacing instances that have no samples. These empty instances can be unregistered, disposed, or alive.
- Next, a **DDS_DataWriter** (p. 485) tries replacing unregistered instances. Since an unregistered instance indicates that the **DDS_DataWriter** (p. 485) 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. 1270) kind.
- Then, a **DDS_DataWriter** (p. 485) tries replacing what is specified by **DDS_DataWriterResourceLimitsQosPolicy::instance_replacement** (p. 1677). 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. 1673)

Enumerator

DDS_UNREGISTERED_INSTANCE_REPLACEMENT	Allows a DDS_DataWriter (p. 485) 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. 1677) [default]

Enumerator

DDS_ALIVE_INSTANCE_REPLACEMENT	Allows a DDS_DataWriter (p. 485) to reclaim alive, acknowledged instances. When an unregistered, acknowledged instance is not available to reclaim, this kind allows a DDS_DataWriter (p. 485) 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 DDS_DataWriter (p. 485) to reclaim disposed acknowledged instances. When an unregistered, acknowledged instance is not available to reclaim, this kind allows a DDS_DataWriter (p. 485) to reclaim a disposed, acknowledged instance. The least recently disposed instance will be reclaimed.
DDS_ALIVE_THEN_DISPOSED_INSTANCE_↔ REPLACEMENT	Allows a DDS_DataWriter (p. 485) 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 DDS_DataWriter (p. 485) 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 DDS_DataWriter (p. 485) 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 DDS_DataWriter (p. 485) 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 DDS_DataWriter (p. 485) to reclaim either an alive acknowledged instance or a disposed acknowledged instance. When an unregistered acknowledged instance is not available to reclaim, this kind allows a DDS_DataWriter (p. 485) to reclaim either an alive, acknowledged instance or a disposed, acknowledged instance. If both instance kinds are available to reclaim, the DDS_DataWriter (p. 485) will reclaim the least recently used (i.e. disposed, registered, or written) instance.

5.148.3 Variable Documentation

5.148.3.1 DDS_DATAWRITERRESOURCELIMITS_QOS_POLICY_NAME

```
const char* const DDS_DATAWRITERRESOURCELIMITS_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_DataWriterResourceLimitsQosPolicy** (p. 1673).

5.149 DATA_WRITER_TRANSFER_MODE

<<**extension**>> (p. 795) Specifies the DataWriter transfer mode QoS.

Data Structures

- struct **DDS_DataWriterShmemRefTransferModeSettings**
Settings related to transferring data using shared memory references.
- struct **DDS_DataWriterTransferModeQosPolicy**
<<**extension**>> (p. 795) Qos related to transferring data

Variables

- const char *const **DDS_DATAWRITERTRANSFERMODE_QOS_POLICY_NAME**
*Stringified human-readable name for **DDS_DataWriterTransferModeQosPolicy** (p. 1683).*

5.149.1 Detailed Description

<<**extension**>> (p. 795) Specifies the DataWriter transfer mode QoS.

5.149.2 Variable Documentation

5.149.2.1 DDS_DATAWRITERTRANSFERMODE_QOS_POLICY_NAME

```
const char* const DDS_DATAWRITERTRANSFERMODE_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_DataWriterTransferModeQosPolicy** (p. 1683).

5.150 DEADLINE

<<**spec**>> (p. 790) Expresses the maximum duration (deadline) within which an instance is expected to be updated.

Data Structures

- struct **DDS_DeadlineQosPolicy**
 <<*spec*>> (p. 790) 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. 1684).

5.150.1 Detailed Description

<<*spec*>> (p. 790) Expresses the maximum duration (deadline) within which an instance is expected to be updated.

5.150.2 Variable Documentation

5.150.2.1 DDS_DEADLINE_QOS_POLICY_NAME

```
const char* const DDS_DEADLINE_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_DeadlineQosPolicy** (p. 1684).

5.151 DESTINATION_ORDER

<<*spec*>> (p. 790) Controls the criteria used to determine the logical order among changes made by **DDS_Publisher** (p. 443) entities to the same instance of data (i.e., matching **DDS_Topic** (p. 175) and key).

Data Structures

- struct **DDS_DestinationOrderQosPolicy**
 <<*spec*>> (p. 790) Controls how the middleware will deal with data sent by multiple **DDS_DataWriter** (p. 485) entities for the same instance of data (i.e., same **DDS_Topic** (p. 175) 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. 1686).

5.151.1 Detailed Description

<<**spec**>> (p. 790) Controls the criteria used to determine the logical order among changes made by **DDS_Publisher** (p. 443) entities to the same instance of data (i.e., matching **DDS_Topic** (p. 175) and key).

5.151.2 Enumeration Type Documentation

5.151.2.1 DDS_DestinationOrderQosPolicyKind

`enum DDS_DestinationOrderQosPolicyKind`

Kinds of destination order.

QoS:

DDS_DestinationOrderQosPolicy (p. 1686)

Enumerator

<code>DDS_BY_RECEPTION_TIMESTAMP_↔ DESTINATIONORDER_QOS</code>	[default] Indicates that data is ordered based on the reception time at each DDS_Subscriber (p. 572). 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.
<code>DDS_BY_SOURCE_TIMESTAMP_↔ DESTINATIONORDER_QOS</code>	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. 1274) and DDS_INSTANCE_SCOPE_↔ DESTINATIONORDER_QOS (p. 1275), then the DDS_BatchQosPolicy::source_timestamp_↔ resolution (p. 1554) and DDS_BatchQosPolicy::thread_safe_write (p. 1555) setting of DDS_BatchQosPolicy (p. 1552) should be set to DDS_DURATION_ZERO (p. 1209) and DDS_BOOLEAN_TRUE (p. 1201) respectively.
	Generated by Doxygen

5.151.2.2 DDS_DestinationOrderQosPolicyScopeKind

enum **DDS_DestinationOrderQosPolicyScopeKind**

Scope of source destination order.

QoS:

DDS_DestinationOrderQosPolicy (p. 1686)

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. 1274). 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. 1274). 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.

5.151.3 Variable Documentation

5.151.3.1 DDS_DESTINATIONORDER_QOS_POLICY_NAME

const char* const DDS_DESTINATIONORDER_QOS_POLICY_NAME [extern]

Stringified human-readable name for **DDS_DestinationOrderQosPolicy** (p. 1686).

5.152 DISCOVERY

<<**extension**>> (p. 795) 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. 1716) and **DDS_DiscoveryQosPolicy::multicast_receive_addresses** (p. 1717) contained in the **DDS_DomainParticipantQos**↔**::discovery** (p. 1729) qos policy.

Data Structures

- struct **DDS_DiscoveryQosPolicy**

<<**extension**>> (p. 795) 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. 1715).

5.152.1 Detailed Description

<<**extension**>> (p. 795) Specifies the attributes required to discover participants in the domain.

5.152.2 Variable Documentation

5.152.2.1 DDS_DISCOVERY_QOS_POLICY_NAME

```
const char* const DDS_DISCOVERY_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_DiscoveryQosPolicy** (p. 1715).

5.153 DISCOVERY_CONFIG

<<**extension**>> (p. 795) Specifies the discovery configuration QoS.

Data Structures

- struct **DDS_BuiltinTopicReaderResourceLimits_t**

<<**extension**>> (p. 795) 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. 1701).
- **#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. 1281) are able to be enabled or disabled by the user.
- **#define DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_DEFAULT** ((`DDS_DiscoveryConfigBuiltinChannelKindMask`) `DDS_DISCOVERYCONFIG_SERVICE_REQUEST_CHANNEL`)
The default value of `DDS_DiscoveryConfigQosPolicy::enabled_builtin_channels` (p. 1701).
- **#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. 1689).*

5.153.1 Detailed Description

<<**extension**>> (p. 795) Specifies the discovery configuration QoS.

5.153.2 Macro Definition Documentation

5.153.2.1 DDS_DISCOVERYCONFIG_BUILTIN_PLUGIN_MASK_NONE

```
#define DDS_DISCOVERYCONFIG_BUILTIN_PLUGIN_MASK_NONE
```

No bits are set.

See also

DDS_DiscoveryConfigBuiltinPluginKindMask (p. 1279)

5.153.2.2 DDS_DISCOVERYCONFIG_BUILTIN_PLUGIN_MASK_DEFAULT

```
#define DDS_DISCOVERYCONFIG_BUILTIN_PLUGIN_MASK_DEFAULT
```

The default value of **DDS_DiscoveryConfigQosPolicy::builtin_discovery_plugins** (p. 1701).

See also

DDS_DiscoveryConfigBuiltinPluginKindMask (p. 1279)

5.153.2.3 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. 1281) are able to be enabled or disabled by the user.

See also

DDS_DiscoveryConfigBuiltinChannelKindMask (p. 1279)

5.153.2.4 DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_DEFAULT

```
#define DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_DEFAULT (( DDS_DiscoveryConfigBuiltinChannelKindMask) DDS_DISCOVERYCONFIG_SERVICE_REQUEST_CHANNEL)
```

The default value of **DDS_DiscoveryConfigQosPolicy::enabled_builtin_channels** (p. 1701).

See also

DDS_DiscoveryConfigBuiltinChannelKindMask (p. 1279)

5.153.2.5 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. 1279)

5.153.3 Typedef Documentation

5.153.3.1 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_DiscoveryConfigBuiltinPluginKind** (p. 1280) values.

QoS:

DDS_DiscoveryConfigQosPolicy (p. 1689)

5.153.3.2 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_DiscoveryConfigBuiltinChannelKind** (p. 1281) values.

QoS:

DDS_DiscoveryConfigQosPolicy (p. 1689)

5.153.4 Enumeration Type Documentation

5.153.4.1 DDS_DiscoveryConfigBuiltinPluginKind

```
enum DDS_DiscoveryConfigBuiltinPluginKind
```

Built-in discovery plugins that can be used.

See also

DDS_DiscoveryConfigBuiltinPluginKindMask (p. 1279)

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. 1279) requires the 'librtildisc' 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 DomainParticipant'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.

5.153.4.2 DDS_DiscoveryConfigBuiltinChannelKind

enum **DDS_DiscoveryConfigBuiltinChannelKind**

Built-in channels that can be enabled.

See also

DDS_DiscoveryConfigBuiltinChannelKindMask (p. 1279)

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.

5.153.4.3 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. 1693)

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. 1692) QoS field. It maintains that contract by writing data to those other participants with a writer that has a DDS_LivelinessQosPolicyKind (p. 1296) of DDS_AUTOMATIC_LIVELINESS_QOS (p. 1297) or DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_↔ QOS (p. 1297) and by asserting itself (at the DDS_DiscoveryConfigQosPolicy::participant_↔ liveliness_assert_period (p. 1692)) 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.

5.153.5 Variable Documentation

5.153.5.1 DDS_DISCOVERYCONFIG_QOS_POLICY_NAME

```
const char* const DDS_DISCOVERYCONFIG_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_DiscoveryConfigQosPolicy** (p. 1689).

5.154 DOMAIN_PARTICIPANT_RESOURCE_LIMITS

<<*extension*>> (p. 795) Various settings that configure how a **DDS_DomainParticipant** (p. 77) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.

Data Structures

- struct **DDS_AllocationSettings_t**
Resource allocation settings.
- struct **DDS_DomainParticipantResourceLimitsQosPolicy**
<<*extension*>> (p. 795) Various settings that configure how a **DDS_DomainParticipant** (p. 77) 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. 1731).

5.154.1 Detailed Description

<<*extension*>> (p. 795) Various settings that configure how a **DDS_DomainParticipant** (p. 77) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.

5.154.2 Enumeration Type Documentation

5.154.2.1 DDS_DomainParticipantResourceLimitsIgnoredEntityReplacementKind

enum **DDS_DomainParticipantResourceLimitsIgnoredEntityReplacementKind**

Available replacement policies for the ignored entities.

QoS:

DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)

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.

5.154.3 Variable Documentation

5.154.3.1 DDS_AUTO_COUNT

```
const DDS_Long DDS_AUTO_COUNT [extern]
```

A special value indicating a quantity that is derived from another QoS value.

5.154.3.2 DDS_DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_NAME

```
const char* const DDS_DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_DomainParticipantResourceLimitsQosPolicy** (p. 1731).

5.155 DURABILITY

<<**spec**>> (p. 790) This QoS policy specifies whether or not RTI Connext will store and deliver previously published data samples to new **DDS_DataReader** (p. 615) entities that join the network later.

Data Structures

- struct **DDS_PersistentStorageSettings**
Configures durable writer history and durable reader state.
- struct **DDS_DurabilityQosPolicy**
<<**spec**>> (p. 790) This QoS policy specifies whether or not RTI Connext will store and deliver previously published data samples to new **DDS_DataReader** (p. 615) 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. 1759).*
- const **DDS_Long** **DDS_AUTO_WRITER_DEPTH**
*A special value used as the default value for **DDS_DurabilityQosPolicy::writer_depth** (p. 1762).*

5.155.1 Detailed Description

<<**spec**>> (p. 790) This QoS policy specifies whether or not RTI Connexx will store and deliver previously published data samples to new **DDS_DataReader** (p. 615) entities that join the network later.

5.155.2 Enumeration Type Documentation

5.155.2.1 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. 1759)

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.

5.155.2.2 DDS_PersistentSynchronizationKind

enum **DDS_PersistentSynchronizationKind**

Determines the level of synchronization with the physical disk.

QoS:

DDS_DurabilityQosPolicy (p. 1759)

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.

5.155.2.3 DDS_DurabilityQosPolicyKind

enum **DDS_DurabilityQosPolicyKind**

Kinds of durability.

QoS:

DDS_DurabilityQosPolicy (p. 1759)

Enumerator

DDS_VOLATILE_DURABILITY_QOS	[default] RTI Connexx does not need to keep any samples of data instances on behalf of any DDS_DataReader (p. 615) that is unknown by the DDS_DataWriter (p. 485) at the time the instance is written. In other words, RTI Connexx 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 Connex will attempt to keep some samples so that they can be delivered to any potential late-joining DDS_DataReader (p. 615). Which particular samples are kept depends on other QoS such as DDS_HistoryQosPolicy (p. 1805) and DDS_ResourceLimitsQosPolicy (p. 1949). RTI Connex is only required to keep the data in memory of the DDS_DataWriter (p. 485) that wrote the data. Data is not required to survive the DDS_DataWriter (p. 485). For this setting to be effective, you must also set the DDS_ReliabilityQosPolicy::kind (p. 1939) to DDS_RELIABLE_RELIABILITY_QOS (p. 1323). This option does not require RTI Persistence Service.
DDS_TRANSIENT_DURABILITY_QOS	RTI Connex will attempt to keep some samples so that they can be delivered to any potential late-joining DDS_DataReader (p. 615). Which particular samples are kept depends on other QoS such as DDS_HistoryQosPolicy (p. 1805) and DDS_ResourceLimitsQosPolicy (p. 1949). RTI Connex is only required to keep the data in memory and not in permanent storage. Data is not tied to the lifecycle of the DDS_DataWriter (p. 485). Data will survive the DDS_DataWriter (p. 485). 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.

5.155.3 Variable Documentation

5.155.3.1 DDS_DURABILITY_QOS_POLICY_NAME

```
const char* const DDS_DURABILITY_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_DurabilityQosPolicy** (p. 1759).

5.155.3.2 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. 1762).

This values resolves to the following:

- **DDS_HistoryQosPolicy::depth** (p. 1807) if the history kind is **DDS_KEEP_LAST_HISTORY_QOS** (p. 1293).
- **DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1952) in the **DDS_ResourceLimitsQosPolicy** (p. 1949) if the history kind is **DDS_KEEP_ALL_HISTORY_QOS** (p. 1293).

5.156 DURABILITY_SERVICE

<<*spec*>> (p. 790) Various settings to configure the external *RTI Persistence Service* used by RTI Connex for DataWriters with a **DDS_DurabilityQosPolicy** (p. 1759) setting of **DDS_PERSISTENT_DURABILITY_QOS** (p. 1288) or **DDS_TRANSIENT_DURABILITY_QOS** (p. 1288).

Data Structures

- struct **DDS_DurabilityServiceQosPolicy**

<<*spec*>> (p. 790) Various settings to configure the external *RTI Persistence Service* used by RTI Connex for DataWriters with a **DDS_DurabilityQosPolicy** (p. 1759) setting of **DDS_PERSISTENT_DURABILITY_QOS** (p. 1288) or **DDS_TRANSIENT_DURABILITY_QOS** (p. 1288).

Variables

- const char *const **DDS_DURABILITYSERVICE_QOS_POLICY_NAME**

Stringified human-readable name for **DDS_DurabilityServiceQosPolicy** (p. 1764).

5.156.1 Detailed Description

<<*spec*>> (p. 790) Various settings to configure the external *RTI Persistence Service* used by RTI Connex for DataWriters with a **DDS_DurabilityQosPolicy** (p. 1759) setting of **DDS_PERSISTENT_DURABILITY_QOS** (p. 1288) or **DDS_TRANSIENT_DURABILITY_QOS** (p. 1288).

5.156.2 Variable Documentation

5.156.2.1 DDS_DURABILITYSERVICE_QOS_POLICY_NAME

```
const char* const DDS_DURABILITYSERVICE_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_DurabilityServiceQosPolicy** (p. 1764).

5.157 ENTITY_FACTORY

<<*spec*>> (p. 790) A QoS policy for all **DDS_Entity** (p. 1360) types that can act as factories for one or more other **DDS_Entity** (p. 1360) types.

Data Structures

- struct **DDS_EntityFactoryQosPolicy**

<<*spec*>> (p. 790) A QoS policy for all **DDS_Entity** (p. 1360) types that can act as factories for one or more other **DDS_Entity** (p. 1360) types.

Variables

- const char *const **DDS_ENTITYFACTORY_QOS_POLICY_NAME**

Stringified human-readable name for **DDS_EntityFactoryQosPolicy** (p. 1788).

5.157.1 Detailed Description

<<*spec*>> (p. 790) A QoS policy for all **DDS_Entity** (p. 1360) types that can act as factories for one or more other **DDS_Entity** (p. 1360) types.

5.157.2 Variable Documentation

5.157.2.1 DDS_ENTITYFACTORY_QOS_POLICY_NAME

```
const char* const DDS_ENTITYFACTORY_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_EntityFactoryQosPolicy** (p. 1788).

5.158 ENTITY_NAME

<<*extension*>> (p. 795) Assigns a name to a **DDS_DomainParticipant** (p. 77), **DDS_Publisher** (p. 443), **DDS_Subscriber** (p. 572), **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615). Except for **DDS_Publisher** (p. 443) and **DDS_Subscriber** (p. 572), these names are visible during the discovery process and in RTI tools to help you visualize and debug your system.

Data Structures

- struct **DDS_EntityNameQosPolicy**

<<*extension*>> (p. 795) Assigns a name and a role name to a **DDS_DomainParticipant** (p. 77), **DDS_Publisher** (p. 443), **DDS_Subscriber** (p. 572), **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615). Except for **DDS_Publisher** (p. 443) and **DDS_Subscriber** (p. 572), 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. 1790).*

5.158.1 Detailed Description

<<**extension**>> (p. 795) Assigns a name to a **DDS_DomainParticipant** (p. 77), **DDS_Publisher** (p. 443), **DDS_Subscriber** (p. 572), **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615). Except for **DDS_Publisher** (p. 443) and **DDS_Subscriber** (p. 572), these names are visible during the discovery process and in RTI tools to help you visualize and debug your system.

5.158.2 Variable Documentation

5.158.2.1 DDS_ENTITYNAME_QOS_POLICY_NAME

```
const char* const DDS_ENTITYNAME_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_EntityNameQosPolicy** (p. 1790).

5.159 EVENT

<<**extension**>> (p. 795) Configures the internal thread in a DomainParticipant that handles timed events.

Data Structures

- `struct DDS_EventQosPolicy`
<<**extension**>> (p. 795) *Settings for event.*

Variables

- `const char *const DDS_EVENT_QOS_POLICY_NAME`
*Stringified human-readable name for **DDS_EventQosPolicy** (p. 1793).*

5.159.1 Detailed Description

<<**extension**>> (p. 795) Configures the internal thread in a DomainParticipant that handles timed events.

5.159.2 Variable Documentation

5.159.2.1 DDS_EVENT_QOS_POLICY_NAME

```
const char* const DDS_EVENT_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_EventQosPolicy** (p. 1793).

5.160 HISTORY

<<*spec*>> (p. 790) Specifies the behavior of RTI Connex 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.

Data Structures

- struct **DDS_HistoryQosPolicy**

<<*spec*>> (p. 790) Specifies the behavior of RTI Connex 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. 1805).

5.160.1 Detailed Description

<<*spec*>> (p. 790) Specifies the behavior of RTI Connex 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.

5.160.2 Enumeration Type Documentation

5.160.2.1 DDS_HistoryQosPolicyKind

```
enum DDS_HistoryQosPolicyKind
```

Kinds of history.

QoS:

DDS_HistoryQosPolicy (p. 1805)

Enumerator

DDS_KEEP_LAST_HISTORY_QOS	[default] Keep the last <code>depth</code> samples. On the publishing side, RTI Connexx will only attempt to keep the most recent <code>depth</code> samples of each instance of data (identified by its key) managed by the DDS_DataWriter (p. 485). 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 DDS_DataReader (p. 615) will only attempt to keep the most recent <code>depth</code> samples received for each instance (identified by its key) until the application takes them via the DDS_DataReader (p. 615) '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 <i>all</i> the samples. On the publishing side, RTI Connexx will attempt to keep all samples (representing each value written) of each instance of data (identified by its key) managed by the DDS_DataWriter (p. 485) until they can be delivered to all subscribers. On the subscribing side, RTI Connexx will attempt to keep all samples of each instance of data (identified by its key) managed by the DDS_DataReader (p. 615). These samples are kept until the application takes them from RTI Connexx via the take() operation.

5.160.3 Variable Documentation

5.160.3.1 DDS_HISTORY_QOS_POLICY_NAME

```
const char* const DDS_HISTORY_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_HistoryQosPolicy** (p. 1805).

5.161 GROUP_DATA

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 165) during discovery.

Data Structures

- struct **DDS_GroupDataQosPolicy**

*Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 165) during discovery.*

Variables

- `const char *const DDS_GROUPDATA_QOS_POLICY_NAME`
*Stringified human-readable name for **DDS_GroupDataQosPolicy** (p. 1803).*

5.161.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 165) during discovery.

5.161.2 Variable Documentation

5.161.2.1 DDS_GROUPDATA_QOS_POLICY_NAME

```
const char* const DDS_GROUPDATA_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_GroupDataQosPolicy** (p. 1803).

5.162 LATENCY_BUDGET

<<**spec**>> (p. 790) 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.

Data Structures

- `struct DDS_LatencyBudgetQosPolicy`
<<**spec**>> (p. 790) 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. 1813).*

5.162.1 Detailed Description

<<**spec**>> (p. 790) 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.

5.162.2 Variable Documentation

5.162.2.1 DDS_LATENCYBUDGET_QOS_POLICY_NAME

```
const char* const DDS_LATENCYBUDGET_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_LatencyBudgetQosPolicy** (p. 1813).

5.163 LIFESPAN

<<*spec*>> (p. 790) Specifies how long the data written by the **DDS_DataWriter** (p. 485) is considered valid.

Data Structures

- struct **DDS_LifespanQosPolicy**

<<*spec*>> (p. 790) Specifies how long the data written by the **DDS_DataWriter** (p. 485) is considered valid.

Variables

- const char *const **DDS_LIFESPAN_QOS_POLICY_NAME**

Stringified human-readable name for **DDS_LifespanQosPolicy** (p. 1815).

5.163.1 Detailed Description

<<*spec*>> (p. 790) Specifies how long the data written by the **DDS_DataWriter** (p. 485) is considered valid.

5.163.2 Variable Documentation

5.163.2.1 DDS_LIFESPAN_QOS_POLICY_NAME

```
const char* const DDS_LIFESPAN_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_LifespanQosPolicy** (p. 1815).

5.164 LIVELINESS

<<*spec*>> (p. 790) Specifies and configures the mechanism that allows **DDS_DataReader** (p. 615) entities to detect when **DDS_DataWriter** (p. 485) entities become disconnected or "dead."

Data Structures

- struct **DDS_LivelinessQosPolicy**

*Specifies and configures the mechanism that allows **DDS_DataReader** (p. 615) entities to detect when **DDS_DataWriter** (p. 485) 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. 1823).*

5.164.1 Detailed Description

<<*spec*>> (p. 790) Specifies and configures the mechanism that allows **DDS_DataReader** (p. 615) entities to detect when **DDS_DataWriter** (p. 485) entities become disconnected or "dead."

5.164.2 Enumeration Type Documentation

5.164.2.1 DDS_LivelinessQosPolicyKind

```
enum DDS_LivelinessQosPolicyKind
```

Kinds of liveliness.

QoS:

DDS_LivelinessQosPolicy (p. 1823)

Enumerator

DDS_AUTOMATIC_LIVELINESS_QOS	[default] The infrastructure will automatically signal liveliness for the DDS_DataWriter (p. 485) (s) at least as often as required by the DDS_DataWriter (p. 485) (S) <code>lease_duration</code> . A DDS_DataWriter (p. 485) with this setting does not need to take any specific action in order to be considered 'alive.' The DDS_DataWriter (p. 485) 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 DDS_DataWriter (p. 485) belonging to the DDS_DomainParticipant (p. 77) (or the DDS_DomainParticipant (p. 77) itself) has asserted its liveliness, then the other DataWriters belonging to that same DDS_DomainParticipant (p. 77) are also alive. The user application takes responsibility to signal liveliness to RTI Connext either by calling DDS_DomainParticipant_assert_liveliness (p. 136), or by calling DDS_DataWriter_assert_liveliness (p. 538), or FooDataWriter_write (p. 496) on any DDS_DataWriter (p. 485) belonging to the DDS_DomainParticipant (p. 77).
DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS	RTI Connext will only assume liveliness of the DDS_DataWriter (p. 485) if the application has asserted liveliness of that DDS_DataWriter (p. 485) itself. The user application takes responsibility to signal liveliness to RTI Connext using the DDS_DataWriter_assert_liveliness (p. 538) function, or by writing some data.

5.164.3 Variable Documentation

5.164.3.1 DDS_LIVELINESS_QOS_POLICY_NAME

```
const char* const DDS_LIVELINESS_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_LivelinessQosPolicy** (p. 1823).

5.165 LOCATORFILTER

<<**extension**>> (p. 795) The QoS policy used to report the configuration of a MultiChannel DataWriter as part of **DDS_PublicationBuiltinTopicData** (p. 1905).

Data Structures

- struct **DDS_LocatorFilter_t**
 <<*extension*>> (p. 795) Specifies the configuration of an individual channel within a *MultiChannel DataWriter*.
- struct **DDS_LocatorFilterSeq**
 Declares IDL *sequence* < **DDS_LocatorFilter_t** (p. 1828) >.
- struct **DDS_LocatorFilterQosPolicy**
 <<*extension*>> (p. 795) The QoS policy used to report the configuration of a *MultiChannel DataWriter* as part of **DDS_PublicationBuiltinTopicData** (p. 1905).

Variables

- const char *const **DDS_LOCATORFILTER_QOS_POLICY_NAME**
 Stringified human-readable name for **DDS_LocatorFilterQosPolicy** (p. 1829).

5.165.1 Detailed Description

<<*extension*>> (p. 795) The QoS policy used to report the configuration of a *MultiChannel DataWriter* as part of **DDS_PublicationBuiltinTopicData** (p. 1905).

5.165.2 Variable Documentation

5.165.2.1 DDS_LOCATORFILTER_QOS_POLICY_NAME

```
const char* const DDS_LOCATORFILTER_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_LocatorFilterQosPolicy** (p. 1829).

5.166 LOGGING

<<*extension*>> (p. 795) Configures the RTI Connext logging facility.

Data Structures

- struct **DDS_LoggingQosPolicy**
 <<*extension*>> (p. 795) Configures the RTI Connext logging facility.

5.166.1 Detailed Description

<<**extension**>> (p. 795) Configures the RTI Connex logging facility.

5.167 MONITORING

<<**extension**>> (p. 795) Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connex telemetry data.

Data Structures

- struct **DDS_MonitoringDedicatedParticipantSettings**
 <<**extension**>> (p. 795) Configures the use of a dedicated **DDS_DomainParticipant** (p. 77) to distribute the RTI Connex application telemetry data.
- struct **DDS_MonitoringEventDistributionSettings**
 <<**extension**>> (p. 795) Configures the distribution of event metrics.
- struct **DDS_MonitoringPeriodicDistributionSettings**
 <<**extension**>> (p. 795) Configures the distribution of periodic metrics.
- struct **DDS_MonitoringLoggingDistributionSettings**
 <<**extension**>> (p. 795) Configures the distribution of log messages.
- struct **DDS_MonitoringDistributionSettings**
 <<**extension**>> (p. 795) Configures the distribution of telemetry data.
- struct **DDS_MonitoringMetricSelection**
 <<**extension**>> (p. 795) 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. 1849) >
- struct **DDS_MonitoringLoggingForwardingSettings**
 <<**extension**>> (p. 795) Configures the forwarding levels of log messages for the different **NDDS_Config_LogFacility** (p. 1446).
- struct **DDS_MonitoringTelemetryData**
 <<**extension**>> (p. 795) Configures the telemetry data that will be distributed.
- struct **DDS_MonitoringQosPolicy**
 <<**extension**>> (p. 795) Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connex telemetry data.

Variables

- const char *const **DDS_MONITORING_QOS_POLICY_NAME**
 Stringified human-readable name for **DDS_MonitoringQosPolicy** (p. 1853).

5.167.1 Detailed Description

<<**extension**>> (p. 795) Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connex telemetry data.

5.167.2 Variable Documentation

5.167.2.1 DDS_MONITORING_QOS_POLICY_NAME

```
const char* const DDS_MONITORING_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_MonitoringQosPolicy** (p. 1853).

5.168 MULTICHANNEL

<<**extension**>> (p. 795) Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.

Data Structures

- struct **DDS_ChannelSettings_t**
 <<**extension**>> (p. 795) Type used to configure the properties of a channel.
- struct **DDS_ChannelSettingsSeq**
 Declares IDL sequence< **DDS_ChannelSettings_t** (p. 1564) >
- struct **DDS_MultiChannelQosPolicy**
 <<**extension**>> (p. 795) 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. 1856).

5.168.1 Detailed Description

<<**extension**>> (p. 795) Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.

5.168.2 Variable Documentation

5.168.2.1 DDS_MULTICHANNEL_QOS_POLICY_NAME

```
const char* const DDS_MULTICHANNEL_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_MultiChannelQosPolicy** (p. 1856).

5.169 OWNERSHIP

<<*spec*>> (p. 790) Specifies whether it is allowed for multiple **DDS_DataWriter** (p. 485) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.

Data Structures

- struct **DDS_OwnershipQosPolicy**

<<*spec*>> (p. 790) Specifies whether it is allowed for multiple **DDS_DataWriter** (p. 485) (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. 1863).*

5.169.1 Detailed Description

<<*spec*>> (p. 790) Specifies whether it is allowed for multiple **DDS_DataWriter** (p. 485) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.

5.169.2 Enumeration Type Documentation

5.169.2.1 DDS_OwnershipQosPolicyKind

```
enum DDS_OwnershipQosPolicyKind
```

Kinds of ownership.

QoS:

DDS_OwnershipQosPolicy (p. 1863)

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. 1301) policy is not specified or supported.
DDS_EXCLUSIVE_OWNERSHIP_QOS	Indicates each instance can only be owned by one DDS_DataWriter (p. 485), but the owner of an instance can change dynamically. The selection of the owner is controlled by the setting of the OWNERSHIP_STRENGTH (p. 1302) policy. The owner is always set to be the highest-strength DDS_DataWriter (p. 485) object among the ones currently active (as determined by the LIVELINESS (p. 1296)).

5.169.3 Variable Documentation

5.169.3.1 DDS_OWNERSHIP_QOS_POLICY_NAME

```
const char* const DDS_OWNERSHIP_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_OwnershipQosPolicy** (p. 1863).

5.170 OWNERSHIP_STRENGTH

<<**spec**>> (p. 790) Specifies the value of the strength used to arbitrate among multiple **DDS_DataWriter** (p. 485) objects that attempt to modify the same instance of a data type (identified by **DDS_Topic** (p. 175) + key).

Data Structures

- struct **DDS_OwnershipStrengthQosPolicy**

*Specifies the value of the strength used to arbitrate among multiple **DDS_DataWriter** (p. 485) objects that attempt to modify the same instance of a data type (identified by **DDS_Topic** (p. 175) + key).*

Variables

- const char *const **DDS_OWNERSHIPSTRENGTH_QOS_POLICY_NAME**

*Stringified human-readable name for **DDS_OwnershipStrengthQosPolicy** (p. 1868).*

5.170.1 Detailed Description

<<*spec*>> (p. 790) Specifies the value of the strength used to arbitrate among multiple **DDS_DataWriter** (p. 485) objects that attempt to modify the same instance of a data type (identified by **DDS_Topic** (p. 175) + key).

5.170.2 Variable Documentation

5.170.2.1 DDS_OWNERSHIPSTRENGTH_QOS_POLICY_NAME

```
const char* const DDS_OWNERSHIPSTRENGTH_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_OwnershipStrengthQosPolicy** (p. 1868).

5.171 PARTITION

<<*spec*>> (p. 790) Set of strings that introduces logical partitions in **DDS_DomainParticipant** (p. 77), **DDS_Publisher** (p. 443), or **DDS_Subscriber** (p. 572) entities.

Data Structures

- struct **DDS_PartitionQosPolicy**

<<*spec*>> (p. 790) Set of strings that introduces logical partitions in **DDS_DomainParticipant** (p. 77), **DDS_Publisher** (p. 443), or **DDS_Subscriber** (p. 572) entities.

Variables

- const char *const **DDS_PARTITION_QOS_POLICY_NAME**

Stringified human-readable name for **DDS_PartitionQosPolicy** (p. 1881).

5.171.1 Detailed Description

<<*spec*>> (p. 790) Set of strings that introduces logical partitions in **DDS_DomainParticipant** (p. 77), **DDS_Publisher** (p. 443), or **DDS_Subscriber** (p. 572) entities.

5.171.2 Variable Documentation

5.171.2.1 DDS_PARTITION_QOS_POLICY_NAME

```
const char* const DDS_PARTITION_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_PartitionQosPolicy** (p. 1881).

5.172 PRESENTATION

<<*spec*>> (p. 790) Specifies how the samples representing changes to data instances are presented to a subscribing application.

Data Structures

- struct **DDS_PresentationQosPolicy**
 <<*spec*>> (p. 790) 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. 1890).

5.172.1 Detailed Description

<<*spec*>> (p. 790) Specifies how the samples representing changes to data instances are presented to a subscribing application.

5.172.2 Enumeration Type Documentation

5.172.2.1 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. 1890)

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 DDS_DataWriter (p. 485), but not across instances in different DDS_DataWriter (p. 485) entities.
DDS_GROUP_PRESENTATION_QOS	Scope spans all instances belonging to DDS_DataWriter (p. 485) entities within the same DDS_Publisher (p. 443).
DDS_HIGHEST_OFFERED_PRESENTATION_QOS	This value only applies to a DDS_Subscriber (p. 572). The DDS_Subscriber (p. 572) will use the access scope specified by each remote DDS_Publisher (p. 443).

5.172.3 Variable Documentation

5.172.3.1 DDS_PRESENTATION_QOS_POLICY_NAME

```
const char* const DDS_PRESENTATION_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_PresentationQosPolicy** (p. 1890).

5.173 PROFILE

<<**extension**>> (p. 795) Configures the way that XML documents containing QoS profiles are loaded by RTI Connex.

Data Structures

- struct **DDS_ProfileQosPolicy**

<<**extension**>> (p. 795) Configures the way that XML documents containing QoS profiles are loaded by RTI Connex.

Variables

- const char *const **DDS_PROFILE_QOS_POLICY_NAME**

Stringified human-readable name for **DDS_ProfileQosPolicy** (p. 1898).

5.173.1 Detailed Description

<<**extension**>> (p. 795) Configures the way that XML documents containing QoS profiles are loaded by RTI Connext.

5.173.2 Variable Documentation

5.173.2.1 DDS_PROFILE_QOS_POLICY_NAME

```
const char* const DDS_PROFILE_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_ProfileQosPolicy** (p. 1898).

5.174 PROPERTY

<<**extension**>> (p. 795) 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.

Data Structures

- struct **DDS_Property_t**
Properties are name/value pairs objects.
- struct **DDS_PropertySeq**
Declares IDL sequence < DDS_Property_t (p. 1900) >
- struct **DDS_PropertyQosPolicy**
<<extension>> (p. 795) 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.

Typedefs

- typedef struct **DDS_PropertyQosPolicy DDS_PropertyQosPolicy**
<<extension>> (p. 795) 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_Long DDS_PropertyQosPolicyHelper_get_number_of_properties** (const struct **DDS_PropertyQosPolicy** *policy)
Gets the number of properties in the input policy.
- DDS_ReturnCode_t DDS_PropertyQosPolicyHelper_assert_property** (struct **DDS_PropertyQosPolicy** *policy, const char *name, const char *value, **DDS_Boolean** propagate)
Asserts the property identified by name in the input policy.
- DDS_ReturnCode_t DDS_PropertyQosPolicyHelper_add_property** (struct **DDS_PropertyQosPolicy** *policy, const char *name, const char *value, **DDS_Boolean** propagate)
Adds a new property to the input policy.
- DDS_ReturnCode_t DDS_PropertyQosPolicyHelper_assert_pointer_property** (struct **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.
- DDS_ReturnCode_t DDS_PropertyQosPolicyHelper_add_pointer_property** (struct **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.
- struct **DDS_Property_t** * **DDS_PropertyQosPolicyHelper_lookup_property** (const struct **DDS_PropertyQosPolicy** *policy, const char *name)
Searches for a property in the input policy given its name.
- struct **DDS_Property_t** * **DDS_PropertyQosPolicyHelper_lookup_property_with_prefix** (const struct **DDS_PropertyQosPolicy** *policy, const char *prefix, const char *name)
Searches for a property in the input policy given its prefix and name.
- DDS_ReturnCode_t DDS_PropertyQosPolicyHelper_remove_property** (struct **DDS_PropertyQosPolicy** *policy, const char *name)
Removes a property from the input policy.
- DDS_ReturnCode_t DDS_PropertyQosPolicyHelper_get_properties** (const struct **DDS_PropertyQosPolicy** *policy, struct **DDS_PropertySeq** *properties, const char *name_prefix)
Retrieves a list of properties whose names match the input prefix.
- DDS_ReturnCode_t DDS_PropertyQosPolicyHelper_get_properties_into_policy** (const struct **DDS_PropertyQosPolicy** *policy, struct **DDS_PropertyQosPolicy** *outPolicy, const char *name_prefix)
This function gets the properties from a policy with a given prefix and sets them into the properties of the output policy.
- DDS_PropertyQosPolicyMutability DDS_PropertyQosPolicyHelper_get_property_mutability** (const char *name, const struct **DDS_PropertyQosPolicy** *policy)
Returns the mutability type of a property.

Variables

- const char *const **DDS_PROPERTY_QOS_POLICY_NAME**
*Stringified human-readable name for **DDS_PropertyQosPolicy** (p. 1901).*

5.174.1 Detailed Description

<<**extension**>> (p. 795) Stores (name, value) pairs that can be used to configure certain parameters of RTI Connex 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 Connex will automatically set some system properties in the **DDS_PropertyQosPolicy** (p. 1901) associated with a **DDS_DomainParticipantQos** (p. 1726). See **System Properties** (p. 970) for additional details.

5.174.2 Typedef Documentation

5.174.2.1 DDS_PropertyQosPolicy

```
typedef struct DDS_PropertyQosPolicy DDS_PropertyQosPolicy
```

<<**extension**>> (p. 795) Stores name/value(string) pairs that can be used to configure certain parameters of RTI Connex 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:

DDS_DomainParticipant (p. 77) **DDS_DataReader** (p. 615) **DDS_DataWriter** (p. 485)

Properties:

RxO (p. 1245) = N/A;
Changeable (p. 1245) = **YES** (p. 1245)

See also

DDS_DomainParticipant_get_builtin_subscriber (p. 129)

5.174.3 Usage

The PROPERTY QoS policy can be used to associate a set of properties in the form of (name, value) pairs with a **DDS_DataReader** (p. 615), **DDS_DataWriter** (p. 485), or **DDS_DomainParticipant** (p. 77). This is similar to the **DDS_UserDataQosPolicy** (p. 2086), 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. 779)
- Durable Reader State, see **Configuring Durable Reader State** (p. 781)
- Builtin Transport Plugins, see **UDIPv4 Transport Property Names in Property QoS Policy of Domain Participant** (p. 1044), **UDIPv6 Transport Property Names in Property QoS Policy of Domain Participant** (p. 1060), and **Shared Memory Transport Property Names in Property QoS Policy of Domain Participant** (p. 1038)
- Extension Transport Plugins, see **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 727)
- **Clock Selection** (p. 25)

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. 2086) or **DDS_GroupDataQoSPolicy** (p. 1803)).

5.174.3.1 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. 1306) page.

5.174.3.2 DDS_PropertyQoSPolicyMutability

```
typedef enum DDS_PropertyQoSPolicyMutability DDS_PropertyQoSPolicyMutability
```

Determines if, and when, the value of a property can change.

5.174.4 Enumeration Type Documentation

5.174.4.1 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.

5.174.5 Function Documentation

5.174.5.1 DDS_PropertyQosPolicyHelper_get_number_of_properties()

```
DDS_Long DDS_PropertyQosPolicyHelper_get_number_of_properties (
    const struct DDS_PropertyQosPolicy * policy )
```

Gets the number of properties in the input policy.

Precondition

policy cannot be NULL.

Parameters

<i>policy</i>	<< <i>in</i> >> (p. 1013) Input policy.

Returns

Number of properties.

5.174.5.2 DDS_PropertyQosPolicyHelper_assert_property()

```
DDS_ReturnCode_t DDS_PropertyQosPolicyHelper_assert_property (
    struct DDS_PropertyQosPolicy * policy,
    const char * name,
    const char * value,
    DDS_Boolean propagate )
```

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

<i>policy</i>	<< <i>in</i> >> (p. 1013) Input policy.
<i>name</i>	<< <i>in</i> >> (p. 1013) Property name.
<i>value</i>	<< <i>in</i> >> (p. 1013) Property value.
<i>propagate</i>	<< <i>in</i> >> (p. 1013) Indicates if the property will be propagated on discovery.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

5.174.5.3 DDS_PropertyQosPolicyHelper_add_property()

```
DDS_ReturnCode_t DDS_PropertyQosPolicyHelper_add_property (
    struct DDS_PropertyQosPolicy * policy,
    const char * name,
    const char * value,
    DDS_Boolean propagate )
```

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 Connex.

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. 1222).

Precondition

policy, name and value cannot be NULL.

The property is not in the policy.

Parameters

<i>policy</i>	<< <i>in</i> >> (p. 1013) Input policy.
<i>name</i>	<< <i>in</i> >> (p. 1013) Property name.
<i>value</i>	<< <i>in</i> >> (p. 1013) Property value.
<i>propagate</i>	<< <i>in</i> >> (p. 1013) Indicates if the property will be propagated on discovery.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222), **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222)

5.174.5.4 DDS_PropertyQosPolicyHelper_assert_pointer_property()

```
DDS_ReturnCode_t DDS_PropertyQosPolicyHelper_assert_pointer_property (
    struct 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.

This is a function similar to **DDS_PropertyQosPolicyHelper_assert_property** (p. 1310). 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

<i>policy</i>	<< in >> (p. 1013) Input policy.
<i>name</i>	<< in >> (p. 1013) Property name.
<i>pointer</i>	<< in >> (p. 1013) The pointer to store in the property.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

5.174.5.5 DDS_PropertyQosPolicyHelper_add_pointer_property()

```
DDS_ReturnCode_t DDS_PropertyQosPolicyHelper_add_pointer_property (
    struct 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.

This is a function similar to **DDS_PropertyQosPolicyHelper_add_property** (p. 1311). 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 Connex.

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. 1222).

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

<i>policy</i>	<< in >> (p. 1013) Input policy.
<i>name</i>	<< in >> (p. 1013) Property name.
<i>pointer</i>	<< in >> (p. 1013) The pointer to store in the property.

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222)

5.174.5.6 DDS_PropertyQosPolicyHelper_lookup_property()

```
struct DDS_Property_t * DDS_PropertyQosPolicyHelper_lookup_property (
    const struct DDS_PropertyQosPolicy * policy,
    const char * name )
```

Searches for a property in the input policy given its name.

Precondition

policy, name and value cannot be NULL.

Parameters

<i>policy</i>	<< <i>in</i> >> (p. 1013) Input policy.
<i>name</i>	<< <i>in</i> >> (p. 1013) Property name.

Returns

The function returns the first property with the given name. If such a property does not exist, the function returns NULL.

5.174.5.7 DDS_PropertyQosPolicyHelper_lookup_property_with_prefix()

```
struct DDS_Property_t * DDS_PropertyQosPolicyHelper_lookup_property_with_prefix (
    const struct DDS_PropertyQosPolicy * policy,
    const char * prefix,
    const char * name )
```

Searches for a property in the input policy given its prefix and name.

Precondition

policy, name and value cannot be NULL.

Parameters

<i>policy</i>	<< <i>in</i> >> (p. 1013) Input policy.
<i>prefix</i>	<< <i>in</i> >> (p. 1013) Property name prefix.
<i>name</i>	<< <i>in</i> >> (p. 1013) Property name.

Returns

The function returns the first property with the given prefix and name. If such a property does not exist, the function returns NULL.

5.174.5.8 DDS_PropertyQosPolicyHelper_remove_property()

```
DDS_ReturnCode_t DDS_PropertyQosPolicyHelper_remove_property (
    struct DDS_PropertyQosPolicy * policy,
    const char * name )
```

Removes a property from the input policy.

If the property does not exist, the function fails with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

Precondition

policy and name cannot be NULL.

The property is in the policy.

Parameters

<i>policy</i>	<< <i>in</i> >> (p. 1013) Input policy.
<i>name</i>	<< <i>in</i> >> (p. 1013) Property name.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222)

5.174.5.9 DDS_PropertyQosPolicyHelper_get_properties()

```
DDS_ReturnCode_t DDS_PropertyQosPolicyHelper_get_properties (
    const struct DDS_PropertyQosPolicy * policy,
    struct DDS_PropertySeq * properties,
    const char * name_prefix )
```

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. 1222).

Precondition

policy, properties and name_prefix cannot be NULL.

Parameters

<i>policy</i>	<< <i>in</i> >> (p. 1013) Input policy.
<i>properties</i>	<< <i>inout</i> >> (p. 1013) A DDS_PropertySeq (p. 1903) object where the set or list of properties will be returned.
<i>name_prefix</i>	Name prefix.

Returns

One of the **Standard Return Codes** (p. 1221), **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222)

5.174.5.10 DDS_PropertyQosPolicyHelper_get_properties_into_policy()

```
DDS_ReturnCode_t DDS_PropertyQosPolicyHelper_get_properties_into_policy (
    const struct DDS_PropertyQosPolicy * policy,
    struct DDS_PropertyQosPolicy * outPolicy,
    const char * name_prefix )
```

This function gets the properties from a policy with a given prefix and sets them into the properties of the output policy.

Precondition

inProperties, outProperties and name_prefix cannot be NULL.

Parameters

<i>policy</i>	<< <i>in</i> >> (p. 1013) Input policy.
<i>outPolicy</i>	<< <i>out</i> >> (p. 1013) Output policy where the properties with the given prefix will be stored. The structure must be initialized.
<i>name_prefix</i>	<< <i>in</i> >> (p. 1013) Name prefix.

Returns

DDS_RETCODE_OK if success, DDS_RETCODE_ERROR otherwise.

5.174.5.11 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. 1310) is returned. Otherwise, it returns the mutability type of the specified property. See **DDS_PropertyQosPolicyMutability** (p. 1309) for more information of the different mutability types.

Parameters

<i>name</i>	<< <i>in</i> >> (p. 1013). The name of the property for which we want to know the mutability type.
<i>policy</i>	<< <i>in</i> >> (p. 1013). 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.

5.174.6 Variable Documentation

5.174.6.1 DDS_PROPERTY_QOS_POLICY_NAME

```
const char* const DDS_PROPERTY_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_PropertyQosPolicy** (p. 1901).

5.175 PUBLISH_MODE

<<**extension**>> (p. 795) Specifies how RTI Connexx sends application data on the network. This QoS policy can be used to tell RTI Connexx to use its *own* thread to send data, instead of the user thread.

Data Structures

- struct **DDS_PublishModeQosPolicy**

<<**extension**>> (p. 795) Specifies how RTI Connexx sends application data on the network. This QoS policy can be used to tell RTI Connexx 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. 1923) and/or **DDS_ChannelSettings_t::priority** (p. 1565).

- #define **DDS_PUBLICATION_PRIORITY_AUTOMATIC**

Constant value for **DDS_PublishModeQosPolicy::priority** (p. 1923) and/or **DDS_ChannelSettings_t::priority** (p. 1565).

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. 1920).

5.175.1 Detailed Description

<<*extension*>> (p. 795) Specifies how RTI Connexx sends application data on the network. This QoS policy can be used to tell RTI Connexx to use its *own* thread to send data, instead of the user thread.

5.175.2 Macro Definition Documentation

5.175.2.1 DDS_PUBLICATION_PRIORITY_UNDEFINED

```
#define DDS_PUBLICATION_PRIORITY_UNDEFINED
```

Initializer value for **DDS_PublishModeQosPolicy::priority** (p. 1923) and/or **DDS_ChannelSettings_t::priority** (p. 1565).

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 multi-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.

5.175.2.2 DDS_PUBLICATION_PRIORITY_AUTOMATIC

```
#define DDS_PUBLICATION_PRIORITY_AUTOMATIC
```

Constant value for **DDS_PublishModeQosPolicy::priority** (p. 1923) and/or **DDS_ChannelSettings_t::priority** (p. 1565).

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.

5.175.3 Enumeration Type Documentation

5.175.3.1 DDS_PublishModeQosPolicyKind

```
enum DDS_PublishModeQosPolicyKind
```

Kinds of publishing mode.

QoS:

DDS_PublishModeQosPolicy (p. 1920)

Enumerator

DDS_SYNCHRONOUS_PUBLISH_MODE_QOS	Indicates to send data synchronously. If DDS_DataWriterProtocolQosPolicy::push_on_write (p. 1650) is DDS_BOOLEAN_TRUE (p. 1201), data is sent immediately in the context of FooDataWriter_write (p. 496). 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. 1650) [default] for DDS_DataWriter (p. 485)
DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS	Indicates to send data asynchronously. Configures the DDS_DataWriter (p. 485) to delegate the task of data transmission to a separate publishing thread. The FooDataWriter_write (p. 496) call does not send the data, but instead schedules the data to be sent later by its associated DDS_Publisher (p. 443). Each DDS_Publisher (p. 443) uses its dedicated publishing thread (DDS_PublisherQos::asynchronous_publisher (p. 1919)) to send data for all its asynchronous DataWriters. For each asynchronous DataWriter, the associated DDS_FlowController (p. 559) determines when the publishing thread is allowed to send the data. DDS_DataWriter_wait_for_asynchronous_publishing (p. 544) and DDS_Publisher_wait_for_asynchronous_publishing (p. 468) enable you to determine when the data has actually been sent. See also DDS_FlowController (p. 559) DDS_HistoryQosPolicy (p. 1805) DDS_DataWriter_wait_for_asynchronous_publishing (p. 544) DDS_Publisher_wait_for_asynchronous_publishing (p. 468) NDDS_Transport_Property_t::gather_send_buffer_count_max (p. 2124)

5.175.4 Variable Documentation

5.175.4.1 DDS_PUBLISHMODE_QOS_POLICY_NAME

```
const char* const DDS_PUBLISHMODE_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_PublishModeQosPolicy** (p. 1920).

5.176 READER_DATA_LIFECYCLE

<<*spec*>> (p. 790) Controls how a DataReader manages the lifecycle of the data that it has received.

Data Structures

- struct **DDS_ReaderDataLifecycleQosPolicy**
 <<*spec*>> (p. 790) 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. 1930).

5.176.1 Detailed Description

<<*spec*>> (p. 790) Controls how a DataReader manages the lifecycle of the data that it has received.

5.176.2 Variable Documentation

5.176.2.1 DDS_READERDATALIFECYCLE_QOS_POLICY_NAME

```
const char* const DDS_READERDATALIFECYCLE_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_ReaderDataLifecycleQosPolicy** (p. 1930).

5.177 RECEIVER_POOL

<<*extension*>> (p. 795) Configures threads used by RTI Connext to receive and process data from transports (for example, UDP sockets).

Data Structures

- struct **DDS_ReceiverPoolQosPolicy**

<<*extension*>> (p. 795) Configures threads used by RTI Connex to receive and process data from transports (for example, UDP sockets).

Variables

- const char *const **DDS_RECEIVERPOOL_QOS_POLICY_NAME**
Stringified human-readable name for **DDS_ReceiverPoolQosPolicy** (p. 1934).
- const **DDS_Long** **DDS_LENGTH_AUTO**
A special value indicating that the actual value will be automatically resolved.

5.177.1 Detailed Description

<<*extension*>> (p. 795) Configures threads used by RTI Connex to receive and process data from transports (for example, UDP sockets).

5.177.2 Variable Documentation

5.177.2.1 DDS_RECEIVERPOOL_QOS_POLICY_NAME

```
const char* const DDS_RECEIVERPOOL_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_ReceiverPoolQosPolicy** (p. 1934).

5.177.2.2 DDS_LENGTH_AUTO

```
const DDS_Long DDS_LENGTH_AUTO [extern]
```

A special value indicating that the actual value will be automatically resolved.

5.178 RELIABILITY

<<*spec*>> (p. 790) Indicates the level of reliability offered/requested by RTI Connex.

Data Structures

- struct **DDS_ReliabilityQosPolicy**
 <<*extension*>> (p. 795) <<*spec*>> (p. 790) 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. 795) *Kinds of acknowledgment.*
- enum **DDS_InstanceStateConsistencyKind** {
DDS_NO_RECOVER_INSTANCE_STATE_CONSISTENCY ,
DDS_RECOVER_INSTANCE_STATE_CONSISTENCY }
 <<*extension*>> (p. 795) *Whether instance state consistency is enabled.*

Variables

- const char *const **DDS_RELIABILITY_QOS_POLICY_NAME**
*Stringified human-readable name for **DDS_ReliabilityQosPolicy** (p. 1936).*

5.178.1 Detailed Description

<<*spec*>> (p. 790) Indicates the level of reliability offered/requested by RTI Connext.

5.178.2 Enumeration Type Documentation

5.178.2.1 DDS_ReliabilityQosPolicyKind

```
enum DDS_ReliabilityQosPolicyKind
```

Kinds of reliability.

QoS:

DDS_ReliabilityQosPolicy (p. 1936)

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 DDS_DataReader (p. 615) and DDS_Topic (p. 175)
DDS_RELIABLE_RELIABILITY_QOS	Specifies RTI Connex will attempt to deliver all samples in its history. Missed samples may be retried. In steady-state (no modifications communicated via the DDS_DataWriter (p. 485)), RTI Connex guarantees that all samples in the DDS_DataWriter (p. 485) history will eventually be delivered to all the DDS_DataReader (p. 615) objects (subject to timeouts that indicate loss of communication with a particular DDS_Subscriber (p. 572)). Outside steady state, the HISTORY (p. 1292) and RESOURCE_LIMITS (p. 1325) policies will determine how samples become part of the history and whether samples can be discarded from it. [default] for DDS_DataWriter (p. 485)

5.178.2.2 DDS_ReliabilityQosPolicyAcknowledgmentModeKind

enum **DDS_ReliabilityQosPolicyAcknowledgmentModeKind**

<<**extension**>> (p. 795) Kinds of acknowledgment.

QoS:

DDS_ReliabilityQosPolicy (p. 1936)

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. 2112) is acknowledged after the subscribing application accesses it, either through calling FooDataReader_take (p. 626) or FooDataReader_read (p. 625) on the DDS sample. If the read or take operation loans the samples, the acknowledgment is done after FooDataReader_return_loan (p. 646) 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 DDS_DataReader (p. 615) are explicitly acknowledged by the subscribing application, after it calls either DDS_DataReader_acknowledge_all (p. 677) or DDS_DataReader_acknowledge_sample (p. 677).

5.178.2.3 DDS_InstanceStateConsistencyKind

enum **DDS_InstanceStateConsistencyKind**

<<**extension**>> (p. 795) Whether instance state consistency is enabled.

QoS:

DDS_ReliabilityQosPolicy (p. 1936)

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 the ServiceRequest 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. 1274) is set to DDS_BY_SOURCE_TIMESTAMP_↔ DESTINATIONORDER_QOS (p. 1274), RECOVER_INSTANCE_STATE_CONSISTENCY can only be used if scope is set to idref_DestinationOrderQosPolicyScopeKind_↔ INSTANCE_SCOPE_DESTINATIONORDER_QOS (the default).

5.178.3 Variable Documentation

5.178.3.1 DDS_RELIABILITY_QOS_POLICY_NAME

```
const char* const DDS_RELIABILITY_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_ReliabilityQosPolicy** (p. 1936).

5.179 RESOURCE_LIMITS

<<*spec*>> (p. 790) 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.

Data Structures

- struct **DDS_ResourceLimitsQosPolicy**

<<*spec*>> (p. 790) 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. 1949).*
- const **DDS_Long** **DDS_LENGTH_UNLIMITED**
A special value indicating an unlimited quantity.

5.179.1 Detailed Description

<<*spec*>> (p. 790) 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.

5.179.2 Variable Documentation

5.179.2.1 DDS_RESOURCELIMITS_QOS_POLICY_NAME

```
const char* const DDS_RESOURCELIMITS_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_ResourceLimitsQosPolicy** (p. 1949).

5.179.2.2 DDS_LENGTH_UNLIMITED

```
const DDS_Long DDS_LENGTH_UNLIMITED [extern]
```

A special value indicating an unlimited quantity.

Examples

```
HelloWorld_subscriber.c.
```

5.180 SERVICE

<<**extension**>> (p. 795) Service QoS policy is used to indicate what kind of service or tool is associated with the DDS entity.

Data Structures

- struct **DDS_ServiceQosPolicy**
 <<**extension**>> (p. 795) Service or tool associated with a DDS entity.

Enumerations

- enum **DDS_ServiceQosPolicyKind** {
 DDS_NO_SERVICE_QOS ,
 DDS_PERSISTENCE_SERVICE_QOS ,
 DDS_QUEUEING_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. 2000).*

5.180.1 Detailed Description

<<**extension**>> (p. 795) Service QoS policy is used to indicate what kind of service or tool is associated with the DDS entity.

5.180.2 Enumeration Type Documentation

5.180.2.1 DDS_ServiceQosPolicyKind

```
enum DDS_ServiceQosPolicyKind
```

Kinds of service.

QoS:

DDS_ServiceQosPolicy (p. 2000)

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_QUEUEING_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.

5.180.3 Variable Documentation

5.180.3.1 DDS_SERVICE_QOS_POLICY_NAME

```
const char* const DDS_SERVICE_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_ServiceQosPolicy** (p. 2000).

5.181 SYSTEM_RESOURCE_LIMITS

<<**extension**>> (p. 795) Configures DomainParticipant-independent resources used by RTI Connex.

Data Structures

- struct **DDS_SystemResourceLimitsQosPolicy**

<<**extension**>> (p. 795) Configures **DDS_DomainParticipant** (p. 77)-independent resources used by RTI Connex. Mainly used to change the maximum number of **DDS_DomainParticipant** (p. 77) 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. 2024).

5.181.1 Detailed Description

<<**extension**>> (p. 795) Configures DomainParticipant-independent resources used by RTI Connex.

5.181.2 Variable Documentation

5.181.2.1 DDS_SYSTEMRESOURCELIMITS_QOS_POLICY_NAME

```
const char* const DDS_SYSTEMRESOURCELIMITS_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_SystemResourceLimitsQosPolicy** (p. 2024).

5.182 TIME_BASED_FILTER

<<*spec*>> (p. 790) Filter that allows a **DDS_DataReader** (p. 615) to specify that it is interested only in (potentially) a subset of the values of the data.

Data Structures

- struct **DDS_TimeBasedFilterQosPolicy**
*<<spec>> (p. 790) Filter that allows a **DDS_DataReader** (p. 615) 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. 2032).*

5.182.1 Detailed Description

<<*spec*>> (p. 790) Filter that allows a **DDS_DataReader** (p. 615) to specify that it is interested only in (potentially) a subset of the values of the data.

5.182.2 Variable Documentation

5.182.2.1 DDS_TIMEBASEDFILTER_QOS_POLICY_NAME

```
const char* const DDS_TIMEBASEDFILTER_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_TimeBasedFilterQosPolicy** (p. 2032).

5.183 TOPIC_DATA

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 165) during discovery.

Data Structures

- struct **DDS_TopicDataQosPolicy**
*Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 165) during discovery.*

Variables

- `const char *const DDS_TOPICDATA_QOS_POLICY_NAME`
*Stringified human-readable name for **DDS_TopicDataQosPolicy** (p. 2040).*

5.183.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 165) during discovery.

5.183.2 Variable Documentation

5.183.2.1 DDS_TOPICDATA_QOS_POLICY_NAME

```
const char* const DDS_TOPICDATA_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_TopicDataQosPolicy** (p. 2040).

5.184 TOPIC_QUERY_DISPATCH

<<**extension**>> (p. 795) Configures the ability of a **DDS_DataWriter** (p. 485) to publish historical samples

Data Structures

- `struct DDS_TopicQueryDispatchQosPolicy`
<<**extension**>> (p. 795) *Configures the ability of a **DDS_DataWriter** (p. 485) to publish samples in response to a **DDS_TopicQuery** (p. 707)*

Variables

- `const char *const DDS_TOPICQUERYDISPATCH_QOS_POLICY_NAME`
*Stringified human-readable name for **DDS_TopicQueryDispatchQosPolicy** (p. 2047).*

5.184.1 Detailed Description

<<**extension**>> (p. 795) Configures the ability of a **DDS_DataWriter** (p. 485) to publish historical samples

5.184.2 Variable Documentation

5.184.2.1 DDS_TOPICQUERYDISPATCH_QOS_POLICY_NAME

```
const char* const DDS_TOPICQUERYDISPATCH_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_TopicQueryDispatchQosPolicy** (p. 2047).

5.185 TRANSPORT_BUILTIN

<<**extension**>> (p. 795) Specifies which built-in transports are used.

Data Structures

- struct **DDS_TransportBuiltinQosPolicy**
 <<**extension**>> (p. 795) 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 **DDS_DomainParticipant** (p. 77) is enabled.
- #define **DDS_TRANSPORTBUILTIN_MASK_DEFAULT**
 The default value of **DDS_TransportBuiltinQosPolicy::mask** (p. 2052).
- #define **DDS_TRANSPORTBUILTIN_MASK_ALL**
 All the available built-in transports are registered automatically when the **DDS_DomainParticipant** (p. 77) is enabled.

Typedefs

- typedef **DDS_Long DDS_TransportBuiltinKindMask**
 A mask of **DDS_TransportBuiltinKind** (p. 1333) 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. 2051).*
- 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".

5.185.1 Detailed Description

<<**extension**>> (p. 795) Specifies which built-in transports are used.

See also

Changing the automatically registered built-in transports (p. 992)

5.185.2 Macro Definition Documentation

5.185.2.1 DDS_TRANSPORTBUILTIN_MASK_NONE

```
#define DDS_TRANSPORTBUILTIN_MASK_NONE
```

None of the built-in transports will be registered automatically when the **DDS_DomainParticipant** (p. 77) is enabled.

The user must explicitly register transports using **NDDS_Transport_Support_register_transport** (p. 730).

See also

DDS_TransportBuiltinKindMask (p. 1333)

5.185.2.2 DDS_TRANSPORTBUILTIN_MASK_DEFAULT

```
#define DDS_TRANSPORTBUILTIN_MASK_DEFAULT
```

The default value of **DDS_TransportBuiltinQosPolicy::mask** (p. 2052).

The set of builtin transport plugins that will be automatically registered with the participant by default. The user can register additional transports using **NDDS_Transport_Support_register_transport** (p. 730).

[default] [UDPv4|Shmem]

See also

DDS_TransportBuiltinKindMask (p. 1333)

5.185.2.3 DDS_TRANSPORTBUILTIN_MASK_ALL

```
#define DDS_TRANSPORTBUILTIN_MASK_ALL
```

All the available built-in transports are registered automatically when the **DDS_DomainParticipant** (p. 77) is enabled.

See also

DDS_TransportBuiltinKindMask (p. 1333)

5.185.3 Typedef Documentation

5.185.3.1 DDS_TransportBuiltinKindMask

```
typedef DDS_Long DDS_TransportBuiltinKindMask
```

A mask of **DDS_TransportBuiltinKind** (p. 1333) bits.

QoS:

DDS_TransportBuiltinQosPolicy (p. 2051)

5.185.4 Enumeration Type Documentation

5.185.4.1 DDS_TransportBuiltinKind

```
enum DDS_TransportBuiltinKind
```

Built-in transport kind.

See also

DDS_TransportBuiltinKindMask (p. 1333)

Enumerator

DDS_TRANSPORTBUILTIN_UDPv4	Built-in UDPv4 transport, :: UDPv4 Transport (p. 1041).
DDS_TRANSPORTBUILTIN_SHMEM	Built-in shared memory transport, :: Shared Memory Transport (p. 1035).
DDS_TRANSPORTBUILTIN_UDPv6	Built-in UDPv6 transport, :: UDPv6 Transport (p. 1058).
DDS_TRANSPORTBUILTIN_UDPv4_WAN	Built-in UDPv4 asymmetric transport, :: Real-Time WAN Transport (p. 1051).

5.185.5 Variable Documentation

5.185.5.1 DDS_TRANSPORTBUILTIN_QOS_POLICY_NAME

```
const char* const DDS_TRANSPORTBUILTIN_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_TransportBuiltinQosPolicy** (p. 2051).

5.185.5.2 DDS_TRANSPORTBUILTIN_SHMEM_ALIAS

```
const char* const DDS_TRANSPORTBUILTIN_SHMEM_ALIAS [extern]
```

Alias name for the shared memory built-in transport: "builtin.shmem".

5.185.5.3 DDS_TRANSPORTBUILTIN_UDPv4_ALIAS

```
const char* const DDS_TRANSPORTBUILTIN_UDPv4_ALIAS [extern]
```

Alias name for the UDPv4 built-in transport: "builtin.udpv4".

5.185.5.4 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".

5.185.5.5 DDS_TRANSPORTBUILTIN_UDPv6_ALIAS

```
const char* const DDS_TRANSPORTBUILTIN_UDPv6_ALIAS [extern]
```

Alias name for the UDPv6 built-in transport: "builtin.udpv6".

5.186 TRANSPORT_MULTICAST

<<*extension*>> (p. 795) Specifies the multicast address on which a **DDS_DataReader** (p. 615) wants to receive its data. It can also specify a port number, as well as a subset of the available (at the **DDS_DomainParticipant** (p. 77) level) transports with which to receive the multicast data.

Modules

- **Multicast Settings**
<<*extension*>> (p. 795) Multicast communication settings.
- **Multicast Mapping**
<<*extension*>> (p. 795) Multicast communication mapping.

Data Structures

- struct **DDS_TransportMulticastQosPolicy**
<<*extension*>> (p. 795) Specifies the multicast address on which a **DDS_DataReader** (p. 615) wants to receive its data. It can also specify a port number as well as a subset of the available (at the **DDS_DomainParticipant** (p. 77) 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. 2058).

5.186.1 Detailed Description

<<*extension*>> (p. 795) Specifies the multicast address on which a **DDS_DataReader** (p. 615) wants to receive its data. It can also specify a port number, as well as a subset of the available (at the **DDS_DomainParticipant** (p. 77) level) transports with which to receive the multicast data.

5.186.2 Enumeration Type Documentation

5.186.2.1 DDS_TransportMulticastQosPolicyKind

enum **DDS_TransportMulticastQosPolicyKind**

Transport Multicast Policy Kind.

See also

DDS_TransportMulticastQosPolicy (p. 2058)

Enumerator

DDS_AUTOMATIC_TRANSPORT_MULTICAST_QOS	Selects the multicast address automatically. NOTE: This setting is required when using the DDS_TransportMulticastMappingQosPolicy (p. 2056).
DDS_UNICAST_ONLY_TRANSPORT_MULTICAST_QOS	Selects a unicast-only mode.

5.186.3 Variable Documentation

5.186.3.1 DDS_TRANSPORTMULTICAST_QOS_POLICY_NAME

const char* const DDS_TRANSPORTMULTICAST_QOS_POLICY_NAME [extern]

Stringified human-readable name for **DDS_TransportMulticastQosPolicy** (p. 2058).

5.187 TRANSPORT_MULTICAST_MAPPING

<<**extension**>> (p. 795) Specifies a list of topic expressions and addresses that can be used by an Entity with a specific topic name to receive data.

Data Structures

- struct **DDS_TransportMulticastMappingQosPolicy**

<<**extension**>> (p. 795) 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. 2056).*

5.187.1 Detailed Description

<<**extension**>> (p. 795) Specifies a list of topic expressions and addresses that can be used by an Entity with a specific topic name to receive data.

5.187.2 Variable Documentation

5.187.2.1 DDS_TRANSPORTMULTICASTMAPPING_QOS_POLICY_NAME

```
const char* const DDS_TRANSPORTMULTICASTMAPPING_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_TransportMulticastMappingQosPolicy** (p. 2056).

5.188 TRANSPORT_PRIORITY

<<**spec**>> (p. 790) This QoS policy allows the application to take advantage of transport that are capable of sending messages with different priorities.

Data Structures

- `struct DDS_TransportPriorityQosPolicy`
<<**spec**>> (p. 790) *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. 2062).*

5.188.1 Detailed Description

<<**spec**>> (p. 790) This QoS policy allows the application to take advantage of transport that are capable of sending messages with different priorities.

5.188.2 Variable Documentation

5.188.2.1 DDS_TRANSPORTPRIORITY_QOS_POLICY_NAME

```
const char* const DDS_TRANSPORTPRIORITY_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_TransportPriorityQosPolicy** (p. 2062).

5.189 TRANSPORT_SELECTION

<<*extension*>> (p. 795) Specifies the physical transports that a **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615) may use to send or receive data.

Data Structures

- struct **DDS_TransportSelectionQosPolicy**
 <<*extension*>> (p. 795) Specifies the physical transports a **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615) may use to send or receive data.

Variables

- const char *const **DDS_TRANSPORTSELECTION_QOS_POLICY_NAME**
 Stringified human-readable name for **DDS_TransportSelectionQosPolicy** (p. 2064).

5.189.1 Detailed Description

<<*extension*>> (p. 795) Specifies the physical transports that a **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615) may use to send or receive data.

5.189.2 Variable Documentation

5.189.2.1 DDS_TRANSPORTSELECTION_QOS_POLICY_NAME

```
const char* const DDS_TRANSPORTSELECTION_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_TransportSelectionQosPolicy** (p. 2064).

5.190 TRANSPORT_UNICAST

<<**extension**>> (p. 795) Specifies a subset of transports and a port number that can be used by an Entity to receive data.

Modules

- **Unicast Settings**
<<**extension**>> (p. 795) *Unicast communication settings.*

Data Structures

- struct **DDS_TransportUnicastQosPolicy**
<<**extension**>> (p. 795) *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. 2065).*

5.190.1 Detailed Description

<<**extension**>> (p. 795) Specifies a subset of transports and a port number that can be used by an Entity to receive data.

5.190.2 Variable Documentation

5.190.2.1 DDS_TRANSPORTUNICAST_QOS_POLICY_NAME

```
const char* const DDS_TRANSPORTUNICAST_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_TransportUnicastQosPolicy** (p. 2065).

5.191 TYPE_CONSISTENCY_ENFORCEMENT

<<**spec**>> (p. 790) Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.

Data Structures

- struct **DDS_TypeConsistencyEnforcementQosPolicy**

<<*spec*>> (p. 790) 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. 2076).*

5.191.1 Detailed Description

<<*spec*>> (p. 790) Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.

5.191.2 Enumeration Type Documentation

5.191.2.1 DDS_TypeConsistencyKind

enum **DDS_TypeConsistencyKind**

Kinds of type consistency.

QoS:

DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)

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 <code>OMG DDS Specification</code> prior to the <code>OMG Extensible and Dynamic Topic Types for DDS Specification</code> .
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 <code>MyDerivedType</code> contains all the members of <code>MyBaseType</code> (<code>member_1</code>) plus some additional elements (<code>member_2</code>). <pre>struct MyBaseType { long member_1; }; struct MyDerivedType: MyBaseType { long member_2; };</pre> Even if <code>MyDerivedType</code> was not explicitly inheriting from <code>MyBaseType</code> the types would still be assignable. For example: <pre>struct MyBaseType { long member_1; }; struct MyDerivedType { long member_1; long member_2; };</pre> For additional information on type assignability refer to the <code>OMG Extensible and Dynamic Topic Types for DDS Specification</code>.
DDS_AUTO_TYPE_COERCION	This AUTO value will be applied as DDS_DISALLOW_TYPE_COERCION (p. 1341) when the data type is annotated with <code>@transfer_mode(SHMEM_REF)</code> and the <code>@language_binding</code> is <code>PLAIN</code> (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. 1341). [default]

5.191.3 Variable Documentation

5.191.3.1 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. 2076).

5.192 TYPESUPPORT

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Allows you to attach application-specific values to a **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615), 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.

Data Structures

- struct **DDS_TypeSupportQosPolicy**

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Allows you to attach application-specific values to a **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615), 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. 2081).

5.192.1 Detailed Description

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Allows you to attach application-specific values to a **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615), 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.

5.192.2 Enumeration Type Documentation

5.192.2.1 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 DDS_DomainParticipant (p. 77) the default behavior is DDS_NOT_SET_CDR_PADDING (p. 1342). When set on a DDS_DataWriter (p. 485) or DDS_DataReader (p. 615) the behavior is to inherit the value from the DDS_DomainParticipant (p. 77).

5.192.3 Variable Documentation

5.192.3.1 DDS_TYPESUPPORT_QOS_POLICY_NAME

```
const char* const DDS_TYPESUPPORT_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_TypeSupportQosPolicy** (p. 2081).

5.193 USER_DATA

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 165) during discovery.

Data Structures

- struct **DDS_UserDataQosPolicy**

*Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 165) during discovery.*

Variables

- const char *const **DDS_USERDATA_QOS_POLICY_NAME**

*Stringified human-readable name for **DDS_UserDataQosPolicy** (p. 2086).*

5.193.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 165) during discovery.

5.193.2 Variable Documentation

5.193.2.1 DDS_USERDATA_QOS_POLICY_NAME

```
const char* const DDS_USERDATA_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_UserDataQosPolicy** (p. 2086).

5.194 WRITER_DATA_LIFECYCLE

<<*spec*>> (p. 790) Controls how a DataWriter handles the lifecycle of the instances (keys) that it is registered to manage.

Data Structures

- struct **DDS_WriterDataLifecycleQosPolicy**
 <<*spec*>> (p. 790) Controls how a **DDS_DataWriter** (p. 485) 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. 2106).

5.194.1 Detailed Description

<<*spec*>> (p. 790) Controls how a DataWriter handles the lifecycle of the instances (keys) that it is registered to manage.

5.194.2 Variable Documentation

5.194.2.1 DDS_WRITERDATALIFECYCLE_QOS_POLICY_NAME

```
const char* const DDS_WRITERDATALIFECYCLE_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_WriterDataLifecycleQosPolicy** (p. 2106).

5.195 WIRE_PROTOCOL

<<*extension*>> (p. 795) Specifies the wire protocol related attributes for the **DDS_DomainParticipant** (p. 77).

Data Structures

- struct **DDS_RtpsWellKnownPorts_t**
 <<*extension*>> (p. 795) RTPS well-known port mapping configuration.
- struct **DDS_WireProtocolQosPolicy**
 <<*extension*>> (p. 795) Specifies the wire-protocol-related attributes for the **DDS_DomainParticipant** (p. 77).

Macros

- **#define DDS_RTPS_RESERVED_PORT_MASK_DEFAULT**
The default value of `DDS_WireProtocolQosPolicy::rtps_reserved_port_mask` (p. 2099).
- **#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. 1347) 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 Connex 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. 2093).

5.195.1 Detailed Description

<<**extension**>> (p. 795) Specifies the wire protocol related attributes for the **DDS_DomainParticipant** (p. 77).

5.195.2 Macro Definition Documentation

5.195.2.1 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. 2099).

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. 1978) 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. 1347)

5.195.2.2 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. 1347)

5.195.2.3 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. 1978) 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 unnecessary 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. 1347)

5.195.3 Typedef Documentation

5.195.3.1 DDS_RtpsReservedPortKindMask

```
typedef DDS_Long DDS_RtpsReservedPortKindMask
```

A mask of **DDS_RtpsReservedPortKind** (p. 1347) bits.

QoS:

DDS_WireProtocolQosPolicy (p. 2093)

5.195.4 Enumeration Type Documentation

5.195.4.1 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. 2099)

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.

5.195.4.2 DDS_WireProtocolQosPolicyAutoKind

```
enum DDS_WireProtocolQosPolicyAutoKind
```

Mechanism to automatically calculate the GUID prefix.

See also

DDS_WireProtocolQosPolicy::rtps_auto_id_kind (p.2100)

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.

5.195.5 Variable Documentation

5.195.5.1 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. 2099) to this value to remain compatible with previous versions of the RTI Connex 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. 1348):

```
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. 1349).

See also

DDS_WireProtocolQosPolicy::rtps_well_known_ports (p. 2099)

DDS_INTEROPERABLE_RTPS_WELL_KNOWN_PORTS (p. 1349)

5.195.5.2 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. 2099) 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. 1349):

```
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 Connex middleware that used fixed port mappings. For backwards compability, please use **DDS_RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS** (p. 1348).

See also

DDS_WireProtocolQosPolicy::rtps_well_known_ports (p. 2099)

DDS_RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS (p. 1348)

5.195.5.3 DDS_WIREPROTOCOL_QOS_POLICY_NAME

```
const char* const DDS_WIREPROTOCOL_QOS_POLICY_NAME [extern]
```

Stringified human-readable name for **DDS_WireProtocolQosPolicy** (p. 2093).

5.196 Extended Qos Support

<<**extension**>> (p. 795) 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.

Data Structures

- struct **DDS_RtpsReliableReaderProtocol_t**

<<**extension**>> (p. 795) QoS related to reliable reader protocol defined in RTPS.

- struct **DDS_RtpsReliableWriterProtocol_t**

<<**extension**>> (p. 795) QoS related to the reliable writer protocol defined in RTPS.

5.196.1 Detailed Description

<<**extension**>> (p. 795) Types and defines used in extended QoS policies.

5.197 Unicast Settings

<<**extension**>> (p. 795) Unicast communication settings.

Data Structures

- struct **DDS_TransportUnicastSettings_t**

Type representing a list of unicast locators.

- struct **DDS_TransportUnicastSettingsSeq**

*Declares IDL sequence < **DDS_TransportUnicastSettings_t** (p. 2067) >*

5.197.1 Detailed Description

<<**extension**>> (p. 795) Unicast communication settings.

5.198 Multicast Settings

<<*extension*>> (p. 795) Multicast communication settings.

Data Structures

- struct **DDS_TransportMulticastSettings_t**
Type representing a list of multicast locators.
- struct **DDS_TransportMulticastSettingsSeq**
Declares IDL sequence< DDS_TransportMulticastSettings_t (p. 2060) >

5.198.1 Detailed Description

<<*extension*>> (p. 795) Multicast communication settings.

5.199 Multicast Mapping

<<*extension*>> (p. 795) Multicast communication mapping.

Data Structures

- struct **DDS_TransportMulticastMappingFunction_t**
Type representing an external mapping function.
- struct **DDS_TransportMulticastMapping_t**
<<extension>> (p. 795) Type representing a list of multicast mapping elements.
- struct **DDS_TransportMulticastMappingSeq**
Declares IDL sequence< DDS_TransportMulticastMapping_t (p. 2053) >

5.199.1 Detailed Description

<<*extension*>> (p. 795) Multicast communication mapping.

5.200 NDDS_DISCOVERY_PEERS

Environment variable or a file that specifies the default values of **DDS_DiscoveryQosPolicy::initial_peers** (p. 1716) and **DDS_DiscoveryQosPolicy::multicast_receive_addresses** (p. 1717) contained in the **DDS_Domain↔ParticipantQos::discovery** (p. 1729) qos policy.

Environment variable or a file that specifies the default values of **DDS_DiscoveryQosPolicy::initial_peers** (p. 1716) and **DDS_DiscoveryQosPolicy::multicast_receive_addresses** (p. 1717) contained in the **DDS_Domain↔ParticipantQos::discovery** (p. 1729) qos policy.

The default value of the **DDS_DomainParticipantQos** (p. 1726) is obtained by calling **DDS_DomainParticipant↔Factory_get_default_participant_qos()** (p. 41).

NDDS_DISCOVERY_PEERS specifies the default value of the **DDS_DiscoveryQosPolicy::initial_peers** (p. 1716) and **DDS_DiscoveryQosPolicy::multicast_receive_addresses** (p. 1717) fields, when the default participant QoS policies have not been explicitly set by the user (i.e., **DDS_DomainParticipantFactory_set_default_participant_qos()** (p. 39) has never been called or was called using **DDS_PARTICIPANT_QOS_DEFAULT** (p. 65)).

If NDDS_DISCOVERY_PEERS does *not* contain a multicast address, then the string sequence **DDS_DiscoveryQos↔Policy::multicast_receive_addresses** (p. 1717) 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. 1717), 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 "udp4://" in front of them (such as udp4://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. 1717).

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. 1353)) or in a file. These formats are described below.

5.200.1 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. 1716) field and the `DDS_DomainParticipant_add_peer()` (p. 149) 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 Connex 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. 1353).

These are separated by the '@' character. The separator may be omitted if a participant ID limit is not explicitly 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).

5.200.1.1 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. 721)) 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. 1354).

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 **DDS_Entity** (p. 1360)) may be used to parse the address string.

5.200.1.2 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. 1717), and **DDS_TransportMulticastSettings_t::receive_address** (p. 2061) 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. 724)).

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. 2123) 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.

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. 2123) 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. 2123) value determines whether the most significant bits (negative sign) or least significant bits (positive sign) are ignored (**Transport Network Address** (p. 724)).

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>udp4://myhostname</element>
    </initial_peers>
  </discovery>
</domain_participant_qos>
```

In this example, the DomainParticipant will attempt to resolve "myhostname" using the UDPv4 transport.

5.200.2 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 "udp4" transport plugin(s) (IPv4)

- udp4://himalaya

Default maximum participant ID on localhost using the "udp4" transport plugin(s) registered at network address FAA0::0

- udp4://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

5.200.3 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

udp4://himalaya            ; 'himalaya' via 'udp4' transport plugin(s)

udp4://FAA0::0#localhost   ; 'localhost' via 'udp4' 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)

```

5.200.4 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.

5.200.5 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://

```

5.200.6 Builtin Transport Class Names

The class names for the builtin transport plugins are:

- `shmem` - **::Shared Memory Transport** (p. 1035)
- `udpv4` - **::UDPv4 Transport** (p. 1041)
- `udpv6` - **::UDPv6 Transport** (p. 1058)

These may be used as the transport names in the **Locator Format** (p. 1353).

5.200.7 NDDS_DISCOVERY_PEERS and Local Host Communication

Suppose you want to communicate with other RTI Connex 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. 2051) of the **DDS_DomainParticipant** (p. 77). 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. 1717)
DDS_DiscoveryQosPolicy::initial_peers (p. 1716)
DDS_DomainParticipant_add_peer() (p. 149)
DDS_PARTICIPANT_QOS_DEFAULT (p. 65)
DDS_DomainParticipantFactory_get_default_participant_qos() (p. 41)
Transport Aliases (p. 721)
Transport Network Address (p. 724)
NDDS_Transport_Support_register_transport() (p. 730)

5.201 Entity Support

DDS_Entity (p. 1360), **DDS_Listener** (p. 1816) and related items.

Data Structures

- struct **DDS_Listener**
<<interface>> (p. 1013) Abstract base class for all Listener interfaces.

Macros

- `#define DDS_Listener_INITIALIZER`
*Initialize the **DDS_Listener::listener_data** (p. 1819) pointer to NULL.*

Typedefs

- typedef struct DDS_EntityImpl **DDS_Entity**
<<interface>> (p. 1013) Abstract base class for all the DDS objects that support QoS policies, a listener, and a status condition.
- typedef struct DDS_DomainEntityImpl **DDS_DomainEntity**
*<<interface>> (p. 1013) Abstract base class for all DDS entities except for the **DDS_DomainParticipant** (p. 77).*

Functions

- **DDS_ReturnCode_t DDS_Entity_enable** (DDS_Entity *self)
*Enables the **DDS_Entity** (p. 1360).*
- **DDS_StatusCondition * DDS_Entity_get_statuscondition** (DDS_Entity *self)
*Allows access to the **DDS_StatusCondition** (p. 1371) associated with the **DDS_Entity** (p. 1360).*
- **DDS_StatusMask DDS_Entity_get_status_changes** (DDS_Entity *self)
*Retrieves the list of communication statuses in the **DDS_Entity** (p. 1360) that are triggered.*
- **DDS_InstanceHandle_t DDS_Entity_get_instance_handle** (const DDS_Entity *self)
*Allows access to the **DDS_InstanceHandle_t** (p. 214) associated with the **DDS_Entity** (p. 1360).*
- **DDS_EntityKind_t DDS_Entity_get_entity_kind** (DDS_Entity *self)
*Allows access to the **DDS_InstanceHandle_t** (p. 214) associated with the **DDS_Entity** (p. 1360).*

5.201.1 Detailed Description

DDS_Entity (p. 1360), **DDS_Listener** (p. 1816) and related items.

DDS_Entity (p. 1360) subtypes are created and destroyed by factory objects. With the exception of **DDS_DomainParticipant** (p. 77), whose factory is **DDS_DomainParticipantFactory** (p. 32), all **DDS_Entity** (p. 1360) factory objects are themselves **DDS_Entity** (p. 1360) subtypes as well.

Important: all **DDS_Entity** (p. 1360) delete operations are inherently thread-unsafe. The user must take extreme care that a given **DDS_Entity** (p. 1360) 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 **DDS_Entity** (p. 1360) is undefined.

5.201.2 Macro Definition Documentation

5.201.2.1 DDS_Listener_INITIALIZER

```
#define DDS_Listener_INITIALIZER
```

Initialize the **DDS_Listener::listener_data** (p. 1819) pointer to NULL.

5.201.3 Typedef Documentation

5.201.3.1 DDS_Entity

```
typedef struct DDS_EntityImpl DDS_Entity
```

<<*interface*>> (p. 1013) 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()`, may return the value **DDS_↵ RETCODE_NOT_ENABLED** (p. 1222).

QoS:

All QoS Policies (p. 1238)

Status:

Status Kinds (p. 1222)

Listener:

DDS_Listener (p. 1816)

5.201.4 Abstract operations

Each derived entity provides the following operations specific to its role in RTI Connext.

5.201.4.1 set_qos (abstract)

This operation sets the QoS policies of the **DDS_Entity** (p. 1360). Each of the derived entity classes provides this operation: **DDS_Entity** (p. 1360) classes (**DDS_DomainParticipant** (p. 77), **DDS_Topic** (p. 175), **DDS_Publisher** (p. 443), **DDS_DataWriter** (p. 485), **DDS_Subscriber** (p. 572), and **DDS_DataReader** (p. 615)) so that the policies that are meaningful to each **DDS_Entity** (p. 1360) can be set. For example, see **DDS_DomainParticipant_set_qos** (p. 151).

Precondition

Certain policies are immutable (see **All QoS Policies** (p. 1238)): they can only be set at **DDS_Entity** (p. 1360) creation time or before the entity is enabled. If `set_qos()` is invoked after the **DDS_Entity** (p. 1360) is enabled and it attempts to change the value of an immutable policy, the operation will fail and return **DDS_↵ RETCODE_IMMUTABLE_POLICY** (p. 1222).

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. 1222).

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. 1222).

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. 1222). In all other cases, none of the policies are modified.

Each derived **DDS_Entity** (p. 1360) class (**DDS_DomainParticipant** (p. 77), **DDS_Topic** (p. 175), **DDS_Publisher** (p. 443), **DDS_DataWriter** (p. 485), **DDS_Subscriber** (p. 572), **DDS_DataReader** (p. 615)) has a corresponding special value of the QoS (**DDS_PARTICIPANT_QOS_DEFAULT** (p. 65), **DDS_PUBLISHER_QOS_DEFAULT** (p. 161), **DDS_SUBSCRIBER_QOS_DEFAULT** (p. 162), **DDS_TOPIC_QOS_DEFAULT** (p. 160), **DDS_DATAWRITER_QOS_DEFAULT** (p. 470), **DDS_DATAREADER_QOS_DEFAULT** (p. 600)). This special value may be used as a parameter to the `set_qos` operation to indicate that the QoS of the **DDS_Entity** (p. 1360) should be changed to match the current default QoS set in the **DDS_Entity** (p. 1360)'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 **DDS_Entity** (p. 1360)'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. 1221) : **DDS_RETCODE_IMMUTABLE_POLICY** (p. 1222), **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222), **DDS_RETCODE_NOT_ALLOWED_BY_SECURITY** (p. 1222).

5.201.4.2 get_qos (abstract)

This operation allows access to the existing set of QoS policies for the **DDS_Entity** (p. 1360). This operation must be provided by each of the derived **DDS_Entity** (p. 1360) classes (**DDS_DomainParticipant** (p. 77), **DDS_Topic** (p. 175), **DDS_Publisher** (p. 443), **DDS_DataWriter** (p. 485), **DDS_Subscriber** (p. 572), and **DDS_DataReader** (p. 615)), so that the policies that are meaningful to each **DDS_Entity** (p. 1360) can be retrieved. For example, see **DDS_DomainParticipant_get_qos** (p. 152).

Possible error codes are **Standard Return Codes** (p. 1221).

5.201.4.3 set_listener (abstract)

This operation installs a **DDS_Listener** (p. 1816) on the **DDS_Entity** (p. 1360). 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 **DDS_Entity** (p. 1360) classes (**DDS_DomainParticipant** (p. 77), **DDS_Topic** (p. 175), **DDS_Publisher** (p. 443), **DDS_DataWriter** (p. 485), **DDS_Subscriber** (p. 572), and **DDS_DataReader** (p. 615)), so that the listener is of the concrete type suitable to the particular **DDS_Entity** (p. 1360).

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 Connex 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. Connex 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 5.941 Effect of Different Combinations of Listeners and Status Bit Masks

	No Bits Set in Mask	Some/All Bits Set in Mask
Listener is Specified	Connex 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	Connex DDS behaves as if the listener is not installed and finds the next most relevant listener for that status	Connex 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 **DDS_Entity** (p.1360). 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.

5.201.4.4 get_listener (abstract)

This operation allows access to the existing **DDS_Listener** (p. 1816) attached to the **DDS_Entity** (p. 1360).

This operation must be provided by each of the derived **DDS_Entity** (p. 1360) classes (**DDS_DomainParticipant** (p. 77), **DDS_Topic** (p. 175), **DDS_Publisher** (p. 443), **DDS_DataWriter** (p. 485), **DDS_Subscriber** (p. 572), and **DDS_DataReader** (p. 615)) so that the listener is of the concrete type suitable to the particular **DDS_Entity** (p. 1360).

If no listener is installed on the **DDS_Entity** (p. 1360), this operation will return NULL.

5.201.4.5 DDS_DomainEntity

```
typedef struct DDS_DomainEntityImpl DDS_DomainEntity
```

<<*interface*>> (p. 1013) Abstract base class for all DDS entities except for the **DDS_DomainParticipant** (p. 77).

Its sole purpose is to *conceptually* express that **DDS_DomainParticipant** (p.77) is a special kind of **DDS_Entity** (p.1360) that acts as a container of all other **DDS_Entity** (p.1360) but itself cannot contain other **DDS_DomainParticipant** (p. 77).

5.201.5 Function Documentation

5.201.5.1 DDS_Entity_enable()

```
DDS_ReturnCode_t DDS_Entity_enable (
    DDS_Entity * self )
```

Enables the **DDS_Entity** (p. 1360).

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. 1289) QoS policy on the corresponding factory for the **DDS_Entity** (p. 1360).

By default, **ENTITY_FACTORY** (p. 1289) is set so that it is not necessary to explicitly call **DDS_Entity_enable** (p. 1363) on newly created entities.

The **DDS_Entity_enable** (p. 1363) operation is idempotent. Calling enable on an already enabled Entity returns OK and has no effect.

If a **DDS_Entity** (p. 1360) 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
- **DDS_Entity_get_statuscondition** (p. 1365)
- 'factory' operations
- **DDS_Entity_get_status_changes** (p. 1365) 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. 1222).

It is legal to delete an **DDS_Entity** (p. 1360) 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. 1788).

Calling enable on an Entity whose factory Entity is not enabled will fail and return **DDS_RETCODE_PRECONDITION**↵
_NOT_MET (p. 1222).

If **DDS_EntityFactoryQosPolicy::autoenable_created_entities** (p. 1789) is TRUE, the enable operation on a factory will automatically enable all entities created from that factory (for example, enabling a **DDS_Publisher** (p. 443) will enable all its contained **DDS_DataWriter** (p. 485) 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`.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

One of the **Standard Return Codes** (p. 1221), **Standard Return Codes** (p. 1221) or **DDS_RETCODE_←PRECONDITION_NOT_MET** (p. 1222).

5.201.5.2 DDS_Entity_get_statuscondition()

```
DDS_StatusCondition * DDS_Entity_get_statuscondition (
    DDS_Entity * self )
```

Allows access to the **DDS_StatusCondition** (p. 1371) associated with the **DDS_Entity** (p. 1360).

The returned condition can then be added to a **DDS_WaitSet** (p. 1371) so that the application can wait for specific status changes that affect the **DDS_Entity** (p. 1360).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

the status condition associated with this entity.

5.201.5.3 DDS_Entity_get_status_changes()

```
DDS_StatusMask DDS_Entity_get_status_changes (
    DDS_Entity * self )
```

Retrieves the list of communication statuses in the **DDS_Entity** (p. 1360) 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()` function.

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.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

list of communication statuses in the **DDS_Entity** (p. 1360) that are triggered.

See also

Status Kinds (p. 1222)

5.201.5.4 DDS_Entity_get_instance_handle()

```
DDS_InstanceHandle_t DDS_Entity_get_instance_handle (
    const DDS_Entity * self )
```

Allows access to the **DDS_InstanceHandle_t** (p. 214) associated with the **DDS_Entity** (p. 1360).

This operation returns the **DDS_InstanceHandle_t** (p. 214) that represents the **DDS_Entity** (p. 1360).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

the instance handle associated with this entity.

5.201.5.5 DDS_Entity_get_entity_kind()

```
DDS_EntityKind_t DDS_Entity_get_entity_kind (
    DDS_Entity * self )
```

Allows access to the **DDS_InstanceHandle_t** (p. 214) associated with the **DDS_Entity** (p. 1360).

This operation returns the **DDS_InstanceHandle_t** (p. 214) that represents the **DDS_Entity** (p. 1360).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

the instance handle associated with this entity.

5.202 Conditions and WaitSets

DDS_Condition (p. 1370) and **DDS_WaitSet** (p. 1371) and related items.

Modules

- **AsyncWaitSet**

<<*extension*>> (p. 1012) A specialization of **DDS_WaitSet** (p. 1371) that provides a mechanism to perform the wait asynchronously and uses a thread pool to dispatch the attached active **DDS_Condition** (p. 1370).

Data Structures

- struct **DDS_ConditionSeq**

Instantiates *FooSeq* (p. 2113) < **DDS_Condition** (p. 1370) >

- struct **DDS_ConditionHandler**

<<*extension*>> (p. 1012) <<*interface*>> (p. 1013) Handler called by the **DDS_Condition_dispatch** (p. 1375).

- struct **DDS_WaitSetProperty_t**

<<*extension*>> (p. 1012) Specifies the **DDS_WaitSet** (p. 1371) behavior for multiple trigger events.

Macros

- #define **DDS_ConditionHandler_INITIALIZER**

<<*experimental*>> (p. 1012) <<*extension*>> (p. 1012) Initializer for new **DDS_ConditionHandler** (p. 1570).

- #define **DDS_WaitSetProperty_t_INITIALIZER**

<<*extension*>> (p. 1012) Initializer for new property instances.

Typedefs

- typedef struct DDS_ConditionImpl **DDS_Condition**

<<*interface*>> (p. 1013) Root class for all the conditions that may be attached to a **DDS_WaitSet** (p. 1371).

- typedef void(* **DDS_ConditionHandler_OnConditionTriggeredCallback**) (void *handler_data, **DDS_Condition** *condition)

Prototype of a **DDS_ConditionHandler** (p. 1570) on_condition_triggered function.

- typedef struct DDS_GuardConditionImpl **DDS_GuardCondition**

<<*interface*>> (p. 1013) A specific **DDS_Condition** (p. 1370) whose trigger_value is completely under the control of the application.

- typedef struct DDS_StatusConditionImpl **DDS_StatusCondition**

<<*interface*>> (p. 1013) A specific **DDS_Condition** (p. 1370) that is associated with each **DDS_Entity** (p. 1360).

- typedef struct DDS_WaitSetImpl **DDS_WaitSet**

<<*interface*>> (p. 1013) Allows an application to wait until one or more of the attached **DDS_Condition** (p. 1370) objects has a trigger_value of **DDS_BOOLEAN_TRUE** (p. 1201) or else until the timeout expires.

Functions

- **DDS_Boolean DDS_Condition_get_trigger_value** (DDS_Condition *self)
Retrieve the trigger_value.
- **DDS_ReturnCode_t DDS_Condition_set_handler** (DDS_Condition *self, const struct DDS_Condition↵
 Handler *handler)
<<extension>> (p. 1012) Registers a DDS_ConditionHandler (p. 1570) in this DDS_Condition (p. 1370).
- struct **DDS_ConditionHandler DDS_Condition_get_handler** (DDS_Condition *self)
<<extension>> (p. 1012) Returns the registered DDS_ConditionHandler (p. 1570).
- void **DDS_Condition_dispatch** (DDS_Condition *self)
*<<extension>> (p. 1012) Calls DDS_ConditionHandler::on_condition_triggered (p. 1571) of the registered DDS↵
 _ConditionHandler (p. 1570).*
- **DDS_Condition * DDS_GuardCondition_as_condition** (DDS_GuardCondition *guardCondition)
Access a DDS_GuardCondition (p. 1370)'s supertype instance.
- **DDS_GuardCondition * DDS_GuardCondition_new** (void)
No argument constructor.
- **DDS_ReturnCode_t DDS_GuardCondition_delete** (DDS_GuardCondition *self)
Destructor.
- **DDS_ReturnCode_t DDS_GuardCondition_set_trigger_value** (DDS_GuardCondition *self, DDS_Boolean
 value)
Set the guard condition trigger value.
- **DDS_Condition * DDS_StatusCondition_as_condition** (DDS_StatusCondition *statusCondition)
Access a DDS_StatusCondition (p. 1371)'s supertype instance.
- **DDS_StatusMask DDS_StatusCondition_get_enabled_statuses** (DDS_StatusCondition *self)
Get the list of statuses enabled on an DDS_Entity (p. 1360).
- **DDS_ReturnCode_t DDS_StatusCondition_set_enabled_statuses** (DDS_StatusCondition *self, DDS_↵
 StatusMask mask)
*This operation defines the list of communication statuses that determine the trigger_value of the DDS_Status↵
 Condition (p. 1371).*
- **DDS_Entity * DDS_StatusCondition_get_entity** (DDS_StatusCondition *self)
Get the DDS_Entity (p. 1360) associated with the DDS_StatusCondition (p. 1371).
- **DDS_WaitSet * DDS_WaitSet_new** (void)
Default no-argument constructor.
- **DDS_WaitSet * DDS_WaitSet_new_ex** (const struct DDS_WaitSetProperty_t *prop)
*<<extension>> (p. 1012) Constructor for a DDS_WaitSet (p. 1371) that may delay for more while specifying that will
 be woken up after the given number of events or delay period, whichever happens first*
- **DDS_ReturnCode_t DDS_WaitSet_delete** (DDS_WaitSet *self)
Destructor.
- **DDS_ReturnCode_t DDS_WaitSet_set_property** (DDS_WaitSet *self, const struct DDS_WaitSetProperty↵
 _t *prop)
*<<extension>> (p. 1012) Sets the DDS_WaitSetProperty_t (p. 2091), to configure the associated DDS_WaitSet
 (p. 1371) to return after one or more trigger events have occurred.*
- **DDS_ReturnCode_t DDS_WaitSet_get_property** (DDS_WaitSet *self, struct DDS_WaitSetProperty↵
 _t *prop)
*<<extension>> (p. 1012) Retrieves the DDS_WaitSetProperty_t (p. 2091) configuration of the associated DDS↵
 WaitSet (p. 1371).*
- **DDS_ReturnCode_t DDS_WaitSet_wait** (DDS_WaitSet *self, struct DDS_ConditionSeq *active_conditions,
 const struct DDS_Duration_t *timeout)
Allows an application thread to wait for the occurrence of certain conditions.

- **DDS_ReturnCode_t DDS_WaitSet_attach_condition** (DDS_WaitSet *self, DDS_Condition *cond)
*Attaches a **DDS_Condition** (p. 1370) to the **DDS_WaitSet** (p. 1371).*
- **DDS_ReturnCode_t DDS_WaitSet_detach_condition** (DDS_WaitSet *self, DDS_Condition *cond)
*Detaches a **DDS_Condition** (p. 1370) from the **DDS_WaitSet** (p. 1371).*
- **DDS_ReturnCode_t DDS_WaitSet_get_conditions** (DDS_WaitSet *self, struct DDS_ConditionSeq *attached_conditions)
*Retrieves the list of attached **DDS_Condition** (p. 1370) (s).*

5.202.1 Detailed Description

DDS_Condition (p. 1370) and **DDS_WaitSet** (p. 1371) and related items.

5.202.2 Macro Definition Documentation

5.202.2.1 DDS_ConditionHandler_INITIALIZER

```
#define DDS_ConditionHandler_INITIALIZER
```

<<**experimental**>> (p. 1012) <<**extension**>> (p. 1012) Initializer for new **DDS_ConditionHandler** (p. 1570).

No memory is allocated. New **DDS_ConditionHandler** (p. 1570) instances stored in the stack should be initialized with this value before they are passed to any functions.

See also

DDS_ConditionHandler (p. 1570)

5.202.2.2 DDS_WaitSetProperty_t_INITIALIZER

```
#define DDS_WaitSetProperty_t_INITIALIZER
```

Value:

```
{ \
    1, DDS_DURATION_INFINITE_VALUE \
}
```

<<**extension**>> (p. 1012) Initializer for new property instances.

Default property specifies `max_event_count = 1` and `max_event_delay = DDS_DURATION_INFINITE`

5.202.3 Typedef Documentation

5.202.3.1 DDS_Condition

```
typedef struct DDS_ConditionImpl DDS_Condition
```

<<*interface*>> (p. 1013) Root class for all the conditions that may be attached to a **DDS_WaitSet** (p. 1371).

This basic class is specialised in three classes:

DDS_GuardCondition (p. 1370), **DDS_StatusCondition** (p. 1371), and **DDS_ReadCondition** (p. 694).

A **DDS_Condition** (p.1370) has a `trigger_value` that can be **DDS_BOOLEAN_TRUE** (p.1201) or **DDS_BOOLEAN_FALSE** (p. 1201) and is set automatically by RTI Connex.

See also

DDS_WaitSet (p. 1371)

5.202.3.2 DDS_ConditionHandler_OnConditionTriggeredCallback

```
typedef void(* DDS_ConditionHandler_OnConditionTriggeredCallback) (void *handler_data, DDS_↔  
Condition *condition)
```

Prototype of a **DDS_ConditionHandler** (p. 1570) `on_condition_triggered` function.

Parameters

<i>handler_data</i>	<< <i>in</i> >> (p. 1013) Data associated with the handler when the handler is set.
<i>condition</i>	st_in associated DDS_Condition (p. 1370) on which the DDS_Condition_dispatch (p. 1375) is called.

5.202.3.3 DDS_GuardCondition

```
typedef struct DDS_GuardConditionImpl DDS_GuardCondition
```

<<*interface*>> (p. 1013) A specific **DDS_Condition** (p. 1370) whose `trigger_value` is completely under the control of the application.

The **DDS_GuardCondition** (p. 1370) provides a way for an application to manually wake up a **DDS_WaitSet** (p. 1371). This is accomplished by attaching the **DDS_GuardCondition** (p. 1370) to the **DDS_WaitSet** (p. 1371) and then setting the `trigger_value` by means of the **DDS_GuardCondition_set_trigger_value** (p. 1377) operation.

See also

DDS_WaitSet (p. 1371)

5.202.3.4 DDS_StatusCondition

```
typedef struct DDS_StatusConditionImpl DDS_StatusCondition
```

<<*interface*>> (p. 1013) A specific **DDS_Condition** (p. 1370) that is associated with each **DDS_Entity** (p. 1360).

The `trigger_value` of the **DDS_StatusCondition** (p. 1371) 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 **DDS_StatusCondition** (p. 1371).

See also

Status Kinds (p. 1222)

DDS_WaitSet (p. 1371), **DDS_Condition** (p. 1370)

DDS_Listener (p. 1816)

5.202.3.5 DDS_WaitSet

```
typedef struct DDS_WaitSetImpl DDS_WaitSet
```

<<*interface*>> (p. 1013) Allows an application to wait until one or more of the attached **DDS_Condition** (p. 1370) objects has a `trigger_value` of **DDS_BOOLEAN_TRUE** (p. 1201) or else until the timeout expires.

5.202.4 Usage

DDS_Condition (p. 1370) (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 **DDS_Condition** (p. 1370) objects (**DDS_StatusCondition** (p. 1371), **DDS_ReadCondition** (p. 694) or **DDS_QueryCondition** (p. 699)) and attaching them to a **DDS_WaitSet** (p. 1371).
- It then waits on that **DDS_WaitSet** (p. 1371) until the `trigger_value` of one or several **DDS_Condition** (p. 1370) objects become **DDS_BOOLEAN_TRUE** (p. 1201).

- It then uses the result of the wait (i.e., `active_conditions`, the list of **DDS_Condition** (p.1370) objects with `trigger_value == DDS_BOOLEAN_TRUE` (p.1201)) to actually get the information:
 - by calling **DDS_Entity_get_status_changes** (p.1365) and then `get_<communication_status>()` on the relevant **DDS_Entity** (p.1360), if the condition is a **DDS_StatusCondition** (p.1371) and the status changes, refer to plain communication status;
 - by calling **DDS_Entity_get_status_changes** (p.1365) and then **DDS_Subscriber_get_datareaders** (p.588) on the relevant **DDS_Subscriber** (p.572) (and then **FooDataReader_read()** (p.625) or **FooDataReader_take** (p.626) on the returned **DDS_DataReader** (p.615) objects), if the condition is a **DDS_StatusCondition** (p.1371) and the status changes refers to **DDS_DATA_ON_READERS_STATUS** (p.1230);
 - by calling **DDS_Entity_get_status_changes** (p.1365) and then **FooDataReader_read()** (p.625) or **FooDataReader_take** (p.626) on the relevant **DDS_DataReader** (p.615), if the condition is a **DDS_StatusCondition** (p.1371) and the status changes refers to **DDS_DATA_AVAILABLE_STATUS** (p.1230);
 - by calling directly **FooDataReader_read_w_condition** (p.630) or **FooDataReader_take_w_condition** (p.632) on a **DDS_DataReader** (p.615) with the **DDS_Condition** (p.1370) as a parameter if it is a **DDS_ReadCondition** (p.694) or a **DDS_QueryCondition** (p.699).

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 **DDS_Condition** (p.1370) objects), **DDS_Condition** (p.1370) objects are meant to embed all that is needed to react properly when enabled. In particular, **DDS_Entity** (p.1360)-related conditions are related to exactly one **DDS_Entity** (p.1360) and cannot be shared.

The blocking behavior of the **DDS_WaitSet** (p.1371) is illustrated below.

The result of a **DDS_WaitSet_wait** (p.1382) operation depends on the state of the **DDS_WaitSet** (p.1371), which in turn depends on whether at least one attached **DDS_Condition** (p.1370) has a `trigger_value` of **DDS_BOOLEAN_TRUE** (p.1201). If the wait operation is called on **DDS_WaitSet** (p.1371) with state **BLOCKED**, it will block the calling thread. If wait is called on a **DDS_WaitSet** (p.1371) with state **UNBLOCKED**, it will return immediately. In addition, when the **DDS_WaitSet** (p.1371) 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 **DDS_Condition** (p.1370).

The **DDS_WaitSet** (p.1371) cannot be used after calling **DDS_DomainParticipantFactory_finalize_instance** (p.38).

5.202.5 Trigger State of a ::DDS_StatusCondition

The `trigger_value` of a **DDS_StatusCondition** (p.1371) is the boolean OR of the `ChangedStatusFlag` of all the communication statuses (see **Status Kinds** (p.1222)) to which it is sensitive. That is, `trigger_value == DDS_BOOLEAN_FALSE` (p.1201) only if all the values of the `ChangedStatusFlags` are **DDS_BOOLEAN_FALSE** (p.1201).

The sensitivity of the **DDS_StatusCondition** (p.1371) to a particular communication status is controlled by the list of `enabled_statuses` set on the condition by means of the **DDS_StatusCondition_set_enabled_statuses** (p.1379) 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 **DDS_StatusCondition** (p.1371) on a **DDS_WaitSet** (p.1371) 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 **DDS_WaitSet_wait** (p.1382).

5.202.6 Trigger State of a ::DDS_ReadCondition

Similar to the **DDS_StatusCondition** (p. 1371), a **DDS_ReadCondition** (p. 694) also has a `trigger_value` that determines whether the attached **DDS_WaitSet** (p. 1371) is BLOCKED or UNBLOCKED. However, unlike the **DDS_StatusCondition** (p. 1371), the `trigger_value` of the **DDS_ReadCondition** (p. 694) is tied to the presence of *at least a sample* managed by RTI Connext with **DDS_SampleStateKind** (p. 710) and **DDS_ViewStateKind** (p. 712) matching those of the **DDS_ReadCondition** (p. 694). Furthermore, for the **DDS_QueryCondition** (p. 699) to have a `trigger_value == DDS_BOOLEAN_TRUE` (p. 1201), the data associated with the sample must be such that the `query_expression` evaluates to **DDS_BOOLEAN_TRUE** (p. 1201).

The fact that the `trigger_value` of a **DDS_ReadCondition** (p. 694) depends on the presence of samples on the associated **DDS_DataReader** (p. 615) implies that a single take operation can potentially change the `trigger_value` of several **DDS_ReadCondition** (p. 694) or **DDS_QueryCondition** (p. 699) conditions. For example, if all samples are taken, any **DDS_ReadCondition** (p. 694) and **DDS_QueryCondition** (p. 699) conditions associated with the **DDS_DataReader** (p. 615) that had their `trigger_value==TRUE` before will see the `trigger_value` change to FALSE. Note that this does not guarantee that **DDS_WaitSet** (p. 1371) 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 **DDS_WaitSet** (p. 1371), the condition transitioning to `trigger_value==FALSE` does not necessarily 'unwake up' the WaitSet as 'unwakening' may not be possible in general.

The consequence is that an application blocked on a **DDS_WaitSet** (p. 1371) 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 **DDS_WaitSet** (p. 1371) objects and taking data associated with the same **DDS_DataReader** (p. 615) entity.

To elaborate further, consider the following example: A **DDS_ReadCondition** (p. 694) that has a `sample_state_mask = {DDS_NOT_READ_SAMPLE_STATE (p. 711)}` will have `trigger_value` of **DDS_BOOLEAN_TRUE** (p. 1201) whenever a new sample arrives and will transition to **DDS_BOOLEAN_FALSE** (p. 1201) 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 **DDS_ReadCondition** (p. 694) had a `sample_state_mask = {DDS_READ_SAMPLE_STATE (p. 711), DDS_NOT_READ_SAMPLE_STATE (p. 711)}`, then the `trigger_value` would only become **DDS_BOOLEAN_FALSE** (p. 1201) 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 **DDS_ReadCondition** (p. 694).

5.202.7 Trigger State of a ::DDS_GuardCondition

The `trigger_value` of a **DDS_GuardCondition** (p. 1370) is completely controlled by the application via the operation **DDS_GuardCondition_set_trigger_value** (p. 1377).

See also

- Status Kinds** (p. 1222)
- DDS_StatusCondition** (p. 1371), **DDS_GuardCondition** (p. 1370)
- DDS_Listener** (p. 1816)

5.202.8 Function Documentation

5.202.8.1 DDS_Condition_get_trigger_value()

```
DDS_Boolean DDS_Condition_get_trigger_value (
    DDS_Condition * self )
```

Retrieve the `trigger_value`.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

the trigger value.

5.202.8.2 DDS_Condition_set_handler()

```
DDS_ReturnCode_t DDS_Condition_set_handler (
    DDS_Condition * self,
    const struct DDS_ConditionHandler * handler )
```

<<**extension**>> (p. 1012) Registers a **DDS_ConditionHandler** (p. 1570) in this **DDS_Condition** (p. 1370).

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 **DDS_Condition_get_handler** (p. 1375) to retrieve the handler.

MT Safety:

It is not safe to call **DDS_Condition_set_handler** (p. 1375), **DDS_Condition_get_handler** (p. 1375) or **DDS_Condition_dispatch** (p. 1375) concurrently.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>handler</i>	The DDS_ConditionHandler (p. 1570) to be called by DDS_Condition_dispatch (p. 1375). If this parameter is null, an no-op handler implementation will be set.

5.202.8.3 DDS_Condition_get_handler()

```
struct DDS_ConditionHandler DDS_Condition_get_handler (
    DDS_Condition * self )
```

<<**extension**>> (p. 1012) Returns the registered **DDS_ConditionHandler** (p. 1570).

If no **DDS_ConditionHandler** (p. 1570) is registered, this operation returns a no-op **DDS_ConditionHandler** (p. 1570) implementation.

5.202.8.4 DDS_Condition_dispatch()

```
void DDS_Condition_dispatch (
    DDS_Condition * self )
```

<<*extension*>> (p. 1012) Calls **DDS_ConditionHandler::on_condition_triggered** (p. 1571) of the registered **DDS_ConditionHandler** (p. 1570).

If the trigger value is true, calling this operation will call the registered **DDS_ConditionHandler** (p. 1570).

If no **DDS_ConditionHandler** (p. 1570) is registered, this operation is a no-op.

5.202.8.5 DDS_GuardCondition_as_condition()

```
DDS_Condition * DDS_GuardCondition_as_condition (
    DDS_GuardCondition * guardCondition )
```

Access a **DDS_GuardCondition** (p. 1370)'s supertype instance.

Parameters

<i>guardCondition</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.202.8.6 DDS_GuardCondition_new()

```
DDS_GuardCondition * DDS_GuardCondition_new (
    void )
```

No argument constructor.

Construct a new guard condition on the heap.

Returns

A new condition with trigger value **DDS_BOOLEAN_FALSE** (p. 1201), or NULL if a condition could not be allocated.

See also

DDS_GuardCondition_delete (p. 1376)

5.202.8.7 DDS_GuardCondition_delete()

```
DDS_ReturnCode_t DDS_GuardCondition_delete (
    DDS_GuardCondition * self )
```

Destructor.

Releases the resources associated with this object.

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this function while another thread may be simultaneously calling **DDS_DomainParticipantFactory_get_instance** (p. 38), **DDS_DomainParticipantFactory_finalize_instance** (p. 38), **DDS_TypeCodeFactory_get_instance** (p. 289), **DDS_GuardCondition_new** (p. 1376), **DDS_WaitSet_new** (p. 1380), **DDS_WaitSet_new_ex** (p. 1380), **DDS_GuardCondition_delete** (p. 1376), **DDS_WaitSet_delete** (p. 1381), **NDDS_Config_Logger_get_instance** (p. 1448), **NDDS_Config_Logger_get_output_device** (p. 1456), **NDDS_Config_Logger_set_output_device** (p. 1457), **DDS_TypeCodeFactory_finalize_instance** (p. 289), **NDDS_Utility_enable_network_capture**, or **NDDS_Utility_disable_network_capture**.

Deleting a NULL condition is safe and has no effect.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.202.8.8 DDS_GuardCondition_set_trigger_value()

```
DDS_ReturnCode_t DDS_GuardCondition_set_trigger_value (
    DDS_GuardCondition * self,
    DDS_Boolean value )
```

Set the guard condition trigger value.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>value</i>	<< <i>in</i> >> (p. 1013) the new trigger value.

5.202.8.9 DDS_StatusCondition_as_condition()

```
DDS_Condition * DDS_StatusCondition_as_condition (
    DDS_StatusCondition * statusCondition )
```

Access a **DDS_StatusCondition** (p. 1371)'s supertype instance.

Parameters

<i>statusCondition</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.202.8.10 DDS_StatusCondition_get_enabled_statuses()

```
DDS_StatusMask DDS_StatusCondition_get_enabled_statuses (
    DDS_StatusCondition * self )
```

Get the list of statuses enabled on an **DDS_Entity** (p. 1360).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

list of enabled statuses.

5.202.8.11 DDS_StatusCondition_set_enabled_statuses()

```
DDS_ReturnCode_t DDS_StatusCondition_set_enabled_statuses (
    DDS_StatusCondition * self,
    DDS_StatusMask mask )
```

This operation defines the list of communication statuses that determine the `trigger_value` of the **DDS_StatusCondition** (p. 1371).

This operation may change the `trigger_value` of the **DDS_StatusCondition** (p. 1371).

DDS_WaitSet (p. 1371) objects' behavior depends on the changes of the `trigger_value` of their attached conditions. Therefore, any **DDS_WaitSet** (p. 1371) to which the **DDS_StatusCondition** (p. 1371) 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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>mask</i>	<< <i>in</i> >> (p. 1013) the list of enables statuses (see Status Kinds (p. 1222))

Returns

One of the **Standard Return Codes** (p. 1221)

5.202.8.12 DDS_StatusCondition_get_entity()

```
DDS_Entity * DDS_StatusCondition_get_entity (
    DDS_StatusCondition * self )
```

Get the **DDS_Entity** (p. 1360) associated with the **DDS_StatusCondition** (p. 1371).

There is exactly one **DDS_Entity** (p. 1360) associated with each **DDS_StatusCondition** (p. 1371).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_Entity (p. 1360) associated with the **DDS_StatusCondition** (p. 1371).

5.202.8.13 DDS_WaitSet_new()

```
DDS_WaitSet * DDS_WaitSet_new (
    void )
```

Default no-argument constructor.

Construct a new **DDS_WaitSet** (p. 1371).

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this function while another thread may be simultaneously calling **DDS_DomainParticipantFactory_get_instance** (p. 38), **DDS_DomainParticipantFactory_finalize_instance** (p. 38), **DDS_TypeCodeFactory_get_instance** (p. 289), **DDS_GuardCondition_new** (p. 1376), **DDS_WaitSet_new** (p. 1380), **DDS_WaitSet_new_ex** (p. 1380), **DDS_GuardCondition_delete** (p. 1376), **DDS_WaitSet_delete** (p. 1381), **NDDDS_Config_Logger_get_instance** (p. 1448), **NDDDS_Config_Logger_get_output_device** (p. 1456), **NDDDS_Config_Logger_set_output_device** (p. 1457), **DDS_TypeCodeFactory_finalize_instance** (p. 289), **NDDDS_Utility_enable_network_capture**, or **NDDDS_Utility_disable_network_capture**.

Returns

A new **DDS_WaitSet** (p. 1371) or NULL if one could not be allocated.

5.202.8.14 DDS_WaitSet_new_ex()

```
DDS_WaitSet * DDS_WaitSet_new_ex (
    const struct DDS_WaitSetProperty_t * prop )
```

<<**extension**>> (p. 1012) Constructor for a **DDS_WaitSet** (p. 1371) 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 **DDS_WaitSet** (p. 1371).

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this function while another thread may be simultaneously calling **DDS_DomainParticipantFactory_get_instance** (p.38), **DDS_DomainParticipantFactory_finalize_instance** (p.38), **DDS_TypeCodeFactory_get_instance** (p.289), **DDS_GuardCondition_new** (p.1376), **DDS_WaitSet_new** (p.1380), **DDS_WaitSet_new_ex** (p.1380), **DDS_GuardCondition_delete** (p.1376), **DDS_WaitSet_delete** (p.1381), **NDDDS_Config_Logger_get_instance** (p.1448), **NDDDS_Config_Logger_get_output_device** (p.1456), **NDDDS_Config_Logger_set_output_device** (p.1457), **DDS_TypeCodeFactory_finalize_instance** (p.289), **NDDDS_Utility_enable_network_capture**, or **NDDDS_Utility_disable_network_capture**.

Returns

A new **DDS_WaitSet** (p. 1371) or NULL if one could not be allocated.

5.202.8.15 DDS_WaitSet_delete()

```
DDS_ReturnCode_t DDS_WaitSet_delete (
    DDS_WaitSet * self )
```

Destructor.

Releases the resources associated with this **DDS_WaitSet** (p. 1371).

Freeing a null pointer is safe and does nothing.

MT Safety:

UNSAFE. It is not safe to delete a **DDS_WaitSet** (p.1371) 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 **DDS_WaitSet_wait** (p.1382). To properly handle this scenario, you can use a **DDS_GuardCondition** (p.1370) to wake up the WaitSet and then wait for the finalization of the thread. In VxWorks, it is unsafe to call this function while another thread may be simultaneously calling **DDS_DomainParticipantFactory_get_instance** (p.38), **DDS_DomainParticipantFactory_finalize_instance** (p.38), **DDS_TypeCodeFactory_get_instance** (p.289), **DDS_GuardCondition_new** (p.1376), **DDS_WaitSet_new** (p.1380), **DDS_WaitSet_new_ex** (p.1380), **DDS_GuardCondition_delete** (p.1376), **DDS_WaitSet_delete** (p.1381), **NDDDS_Config_Logger_get_instance** (p.1448), **NDDDS_Config_Logger_get_output_device** (p.1456), **NDDDS_Config_Logger_set_output_device** (p.1457), **DDS_TypeCodeFactory_finalize_instance** (p.289), **NDDDS_Utility_enable_network_capture**, or **NDDDS_Utility_disable_network_capture**.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

5.202.8.16 DDS_WaitSet_set_property()

```
DDS_ReturnCode_t DDS_WaitSet_set_property (
    DDS_WaitSet * self,
    const struct DDS_WaitSetProperty_t * prop )
```

<<*extension*>> (p. 1012) Sets the **DDS_WaitSetProperty_t** (p.2091), to configure the associated **DDS_WaitSet** (p. 1371) to return after one or more trigger events have occurred.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>prop</i>	<< <i>in</i> >> (p. 1013)

Returns

One of the **Standard Return Codes** (p. 1221)

5.202.8.17 DDS_WaitSet_get_property()

```
DDS_ReturnCode_t DDS_WaitSet_get_property (
    DDS_WaitSet * self,
    struct DDS_WaitSetProperty_t * prop )
```

<<*extension*>> (p. 1012) Retrieves the **DDS_WaitSetProperty_t** (p.2091) configuration of the associated **DDS_WaitSet** (p. 1371).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>prop</i>	<< <i>out</i> >> (p. 1013)

Returns

One of the **Standard Return Codes** (p. 1221)

5.202.8.18 DDS_WaitSet_wait()

```

DDS_ReturnCode_t DDS_WaitSet_wait (
    DDS_WaitSet * self,
    struct DDS_ConditionSeq * active_conditions,
    const struct DDS_Duration_t * timeout )

```

Allows an application thread to wait for the occurrence of certain conditions.

If none of the conditions attached to the **DDS_WaitSet** (p. 1371) have a `trigger_value` of **DDS_BOOLEAN_TRUE** (p. 1201), 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. 1201) (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 **DDS_Condition** (p. 1370) objects are **DDS_BOOLEAN_TRUE** (p. 1201), wait fails with **DDS_RETCODE_TIMEOUT** (p. 1222). 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. 1222).

Note: The resolution of the `timeout` period is constrained by the resolution of the system clock.

When the **DDS_WaitSet** (p. 1371) 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 **DDS_WaitSet** (p. 1371). If the wait operation is invoked on a **DDS_WaitSet** (p. 1371) that already has a thread blocking on it, the operation will return immediately with the value **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

Parameters

<i>active_conditions</i>	<< <i>inout</i> >> (p. 1013) a valid non-NULL DDS_ConditionSeq (p. 1571) object. Note that RTI Connext will not allocate a new object if <code>active_conditions</code> is NULL; the function will return DDS_RETCODE_PRECONDITION_NOT_MET (p. 1222).
<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>timeout</i>	<< <i>in</i> >> (p. 1013) a wait timeout

Returns

One of the **Standard Return Codes** (p. 1221) or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222) or **DDS_RETCODE_TIMEOUT** (p. 1222).

5.202.8.19 DDS_WaitSet_attach_condition()

```

DDS_ReturnCode_t DDS_WaitSet_attach_condition (
    DDS_WaitSet * self,
    DDS_Condition * cond )

```

Attaches a **DDS_Condition** (p. 1370) to the **DDS_WaitSet** (p. 1371).

It is possible to attach a **DDS_Condition** (p. 1370) on a **DDS_WaitSet** (p. 1371) that is currently being waited upon (via the wait operation). In this case, if the **DDS_Condition** (p. 1370) has a `trigger_value` of **DDS_BOOLEAN_TRUE** (p. 1201), then attaching the condition will unblock the **DDS_WaitSet** (p. 1371).

If the same **DDS_Condition** (p. 1370) is attached multiple times, only the first attachment has any effect. Subsequent attempts to attach the same **DDS_Condition** (p. 1370) are silently ignored. As a result, the **DDS_Condition** (p. 1370) is only considered attached once and must only be detached once.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>cond</i>	<< <i>in</i> >> (p. 1013) Condition to be attached.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

5.202.8.20 DDS_WaitSet_detach_condition()

```
DDS_ReturnCode_t DDS_WaitSet_detach_condition (
    DDS_WaitSet * self,
    DDS_Condition * cond )
```

Detaches a **DDS_Condition** (p. 1370) from the **DDS_WaitSet** (p. 1371).

It is possible to detach a **DDS_Condition** (p. 1370) on a **DDS_WaitSet** (p. 1371) that is currently being waited upon (via the wait operation). If the **DDS_Condition** (p. 1370) was not attached to the **DDS_WaitSet** (p. 1371), the operation will return **DDS_RETCODE_BAD_PARAMETER** (p. 1222).

If the same **DDS_Condition** (p. 1370) had been previously attached multiple times, only a single detach is required; the **DDS_Condition** (p. 1370) is considered attached only once.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>cond</i>	<< <i>in</i> >> (p. 1013) Condition to be detached.

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222).

5.202.8.21 DDS_WaitSet_get_conditions()

```
DDS_ReturnCode_t DDS_WaitSet_get_conditions (
    DDS_WaitSet * self,
    struct DDS_ConditionSeq * attached_conditions )
```

Retrieves the list of attached **DDS_Condition** (p. 1370) (s).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>attached_conditions</i>	<< <i>inout</i> >> (p. 1013) a DDS_ConditionSeq (p. 1571) object where the list of attached conditions will be returned

Returns

One of the **Standard Return Codes** (p. 1221), or **DDS_RETCODE_OUT_OF_RESOURCES** (p. 1222).

5.203 Cookie

<<*extension*>> (p. 1012) Unique identifier for a written data sample

Data Structures

- struct **DDS_Cookie_t**
 <<*extension*>> (p. 1012) Sequence of bytes.
- struct **DDS_CookieSeq**
 Declares IDL sequence < **DDS_Cookie_t** (p. 1580) > .

Functions

- **DDS_Boolean DDS_Cookie_from_pointer** (struct **DDS_Cookie_t** *self, const void *pointer)
 Stores a pointer in a cookie.
- void * **DDS_Cookie_to_pointer** (const struct **DDS_Cookie_t** *self)
 Returns the pointer previously stored in the cookie.

5.203.1 Detailed Description

<<*extension*>> (p. 1012) Unique identifier for a written data sample

5.203.2 Function Documentation

5.203.2.1 DDS_Cookie_from_pointer()

```
DDS_Boolean DDS_Cookie_from_pointer (
    struct   DDS_Cookie_t * self,
    const void * pointer )
```

Stores a pointer in a cookie.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>pointer</i>	<< <i>in</i> >> (p. 1013) The pointer to be stored in the cookie. Can be NULL.

Returns

DDS_BOOLEAN_TRUE if the cookie was successfully filled with the pointer, DDS_BOOLEAN_FALSE otherwise.

5.203.2.2 DDS_Cookie_to_pointer()

```
void * DDS_Cookie_to_pointer (
    const struct   DDS_Cookie_t * self )
```

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.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

The pointer stored in the cookie.

5.204 Sample Flags

<<*extension*>> (p. 1012) 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.

5.204.1 Detailed Description

<<*extension*>> (p. 1012) Flags for samples.

5.204.2 Typedef Documentation

5.204.2.1 DDS_SampleFlagBits

```
typedef enum DDS_SampleFlagBits DDS_SampleFlagBits
```

Type to identify the sample flags reserved by RTI.

5.204.2.2 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

5.204.3 Enumeration Type Documentation

5.204.3.1 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 Connex 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.
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 DDS_TopicQuery (p. 707). When this bit is not set and DDS_SampleInfo::topic_query_guid (p. 1995) is different from DDS_GUID_UNKNOWN (p. 1213), this sample is the last sample for that TopicQuery coming from the DataWriter identified by DDS_SampleInfo::source_guid (p. 1994).
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. 1293) DDS_HistoryQosPolicy (p. 1805) QoS or because the duration in DDS_LifespanQosPolicy (p. 1815) 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.
	Generated by Doxygen

5.205 WriteParams

<<*extension*>> (p. 1012)

Data Structures

- 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. 1012) Input parameters for writing with **FooDataWriter_write_w_params** (p. 501), **FooDataWriter_dispose_w_params** (p. 504), **FooDataWriter_register_instance_w_params** (p. 493), **FooDataWriter_unregister_instance_w_params** (p. 496)

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. 2101).*

5.205.1 Detailed Description

<<*extension*>> (p. 1012)

5.205.2 Function Documentation

5.205.2.1 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

<i>self</i>	<< <i>in</i> >> (p. 1013) This sample identity.
<i>other</i>	<< <i>in</i> >> (p. 1013) The other sample identity to be compared with this sample identity.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the two sample identities have equal values, or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.205.2.2 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. 2102) is enabled and the same params instance is used in multiple calls to **FooDataWriter_write_w_params** (p. 501)

5.205.3 Variable Documentation

5.205.3.1 writer_guid

```
struct DDS_GUID_t DDS_SampleIdentity_t::writer_guid
```

16-byte identifier identifying the virtual GUID.

5.205.3.2 sequence_number

```
struct DDS_SequenceNumber_t DDS_SampleIdentity_t::sequence_number
```

monotonically increasing 64-bit integer that identifies the sample in the data source.

5.205.3.3 DDS_AUTO_SAMPLE_IDENTITY

```
const struct DDS_SampleIdentity_t DDS_AUTO_SAMPLE_IDENTITY [extern]
```

The AUTO sample identity.

Special **DDS_AUTO_SAMPLE_IDENTITY** (p. 1391) value {**DDS_GUID_AUTO** (p. 1213), **DDS_AUTO_SEQUENCE_↵**
NUMBER (p. 1218)}

5.205.3.4 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. 1392) value {**DDS_GUID_UNKNOWN** (p. 1213), **DDS_↵**
SEQUENCE_NUMBER_UNKNOWN (p. 1218)}

5.205.3.5 DDS_WRITEPARAMS_DEFAULT

```
const struct DDS_WriteParams_t DDS_WRITEPARAMS_DEFAULT [extern]
```

Initializer for **DDS_WriteParams_t** (p. 2101).

5.206 Heap Support in C

<<*extension*>> (p. 1012) Heap allocation and free routines in C

Functions

- void * **DDS_Heap_malloc** (size_t numElem, size_t size)
Performs the logical equivalent of calloc().
- void * **DDS_Heap_malloc** (size_t size)
Performs the logical equivalent of malloc().
- void **DDS_Heap_free** (void *ptr)
Performs the logical equivalent of free().

5.206.1 Detailed Description

<<*extension*>> (p. 1012) Heap allocation and free routines in C

The functions in this class ensure consistent cross-platform implementations for memory allocation (**DDS_Heap_↵**
malloc() (p. 1393) and **DDS_Heap_malloc()** (p. 1393)) and deletion (**DDS_Heap_free()** (p. 1393)).

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. 1393)/ **DDS_Heap_malloc()** (p. 1393); to release an optional member previously reserved by **FootTypeSupport_create_data_w_params()** (p. 216), use **DDS_Heap_free()** (p. 1393).

5.206.2 Function Documentation

5.206.2.1 DDS_Heap_calloc()

```
void * DDS_Heap_calloc (
    size_t numElem,
    size_t size )
```

Performs 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

<i>numElem</i>	<< <i>in</i> >> (p. 1013) The number of elements in the array to be allocated
<i>size</i>	<< <i>in</i> >> (p. 1013) The size in bytes of each element in the array

Returns

If insufficient memory is available, this function will return NULL. Otherwise, upon success it returns a pointer to the allocated space.

5.206.2.2 DDS_Heap_malloc()

```
void * DDS_Heap_malloc (
    size_t size )
```

Performs the logical equivalent of malloc().

Allocates unused space for an object of the specified `size`. The memory will be initialized to zeros.

Parameters

<i>size</i>	<< <i>in</i> >> (p. 1013) The size in bytes of the object to be allocated

Returns

If insufficient memory is available, this function will return NULL. Otherwise, upon success it returns a pointer to the allocated space.

5.206.2.3 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

<i>ptr</i>	<< <i>in</i> >> (p. 1013) A pointer to the space to be deallocated

5.207 Builtin Qos Profiles

<<*extension*>> (p. 1012) 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 Connex Micro.
- const char *const **DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_2_4_9**
Sets the values necessary to communicate with RTI Connex 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 Connex Micro versions 2.4.3 and earlier.
- 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 Connex 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 shm 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_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 Connex 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 LIVENESS QoS to be compatible with RTI Connex Micro 2.4.3.

- const char *const **DDS_SNIPPET_COMPATIBILITY_OTHER_DDS_VENDORS_ENABLE**

QoS Snippet that configures RTI Connex to interoperate with other DDS vendors.

- const char *const **DDS_SNIPPET_5_1_0_TRANSPORT_ENABLE**

QoS Snippet that configures RTI Connex to interoperate with RTI Connex 5.1.0 and below for UDPv6 and SHMEM transports.

5.207.1 Detailed Description

<<**extension**>> (p. 1012) QoS libraries, profiles, and snippets that are automatically built into RTI Connex.

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 Connex 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 Connex version, use the BuiltinQosLib::Baseline profile.
- Generic.X: These profiles allow you to easily configure different features and communication use-cases with RTI Connex. 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 **DDS_DataWriter** (p. 485) 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 Connex installation.

- BaselineRoot.documentationONLY.xml contains the root baseline QoS profile that corresponds to the default values of RTI Connex 5.0.0.
- BuiltinProfiles.documentationONLY.xml contains the rest of the built-in QoS Profiles and QoS Snippets.

5.207.2 Variable Documentation

5.207.2.1 DDS_BUILTIN_QOS_LIB

```
const char* const DDS_BUILTIN_QOS_LIB [extern]
```

A library of non-experimental QoS profiles.

String-version: "BuiltinQosLib"

5.207.2.2 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 Connex 5.0.0.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400)

String-version: "Baseline.Root"

5.207.2.3 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 Connex version is released – without changing your application code.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400)

String-version: "Baseline"

5.207.2.4 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. 1400)

String-version: "Baseline.5.0.0"

5.207.2.5 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. 1400)

String-version: "Baseline.5.1.0"

5.207.2.6 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. 1400)

String-version: "Baseline.5.2.0"

5.207.2.7 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. 1400)

String-version: "Baseline.5.3.0"

5.207.2.8 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. 1400)

String-version: "Baseline.6.0.0"

5.207.2.9 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. 1400)

String-version: "Baseline.6.1.0"

5.207.2.10 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. 1400)

String-version: "Baseline.7.0.0"

5.207.2.11 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. 1400)

String-version: "Baseline.7.1.0"

5.207.2.12 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. 1400)

String-version: "Generic.Common"

5.207.2.13 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. 1400)

String-version: "Generic.Monitoring.Common"

5.207.2.14 DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY

```
const char* const DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY [extern]
```

Sets the values necessary to communicate with RTI Connex Micro.

This profile will always represent the QoS values required for interoperability between the most recent version of RTI Connex Micro at the time of release of the most recent version of RTI Connex.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400)

String-version: "Generic.ConnexMicroCompatibility"

5.207.2.15 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 Connex Micro versions 2.4.4 through at least 2.4.9.

At the time of the release of RTI Connex 5.3.0 it was not necessary to set any QoS values in order to interoperate with RTI Connex Micro. This applies to RTI Connex Micro versions 2.4.4 and later. The most recent version of RTI Connex Micro at the time of release of RTI Connex 5.3.0 was 2.4.9. There is no guarantee that this profile will interoperate with versions of RTI Connex Micro after 2.4.9.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400)

String-version: "Generic.ConnexMicroCompatibility.2.4.9"

5.207.2.16 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 Connex Micro versions 2.4.3 and earlier.

RTI Connex Micro versions 2.4.3 and earlier only supported the **DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS** (p. 1297) LivelinessQos kind. In order to be compatible with these versions of RTI Connex Micro, the **DDS_DataReader** (p. 615) and **DDS_DataWriter** (p. 485) must have their Liveliness kind changed to this value because the default kind in RTI Connex is **DDS_AUTOMATIC_LIVELINESS_QOS** (p. 1297).

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400)

String-version: "Generic.ConnexMicroCompatibility.2.4.3"

5.207.2.17 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. 1400)

String-version: "Generic.OtherDDSVendorCompatibility"

5.207.2.18 DDS_PROFILE_GENERIC_510_TRANSPORT_COMPATIBILITY

```
const char* const DDS_PROFILE_GENERIC_510_TRANSPORT_COMPATIBILITY [extern]
```

Sets the values necessary to interoperate with RTI Connex 5.1.0 using the UDPv6 and/or SHMEM transports.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400)

String-version: "Generic.510TransportCompatibility"

5.207.2.19 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"

5.207.2.20 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. 1323) for **DDS_ReliabilityQosPolicy** (p. 1936) with a **DDS_KEEP_ALL_HISTORY_QOS** (p. 1293) for the **DDS_HistoryQosPolicyKind** (p. 1292).

This QoS Profile also optimizes the reliability protocol setting for this configuration.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Generic.StrictReliable"

5.207.2.21 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. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Generic.KeepLastReliable"

5.207.2.22 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. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Generic.BestEffort"

5.207.2.23 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. 1404) 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. 1404) QoS Profile..

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Generic.StrictReliable.HighThroughput"

5.207.2.24 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. 1404) 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. 1404) QoS Profile.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Generic.StrictReliable.LowLatency"

5.207.2.25 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. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Generic.Participant.LargeData"

5.207.2.26 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. 1425)

Enable Large Data optimizations

- **DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LARGE_DATA** (p. 1417)
- **DDS_SNIPPET_OPTIMIZATION_DATACACHE_LARGE_DATA_DYNAMICMEMALLOC** (p. 1418) In Library↔
: **DDS_BUILTIN_QOS_LIB** (p. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Generic.Participant.LargeData.Monitoring"

See also

DDS_PROFILE_GENERIC_PARTICIPANT_LARGE_DATA (p. 1405)

DDS_PROFILE_GENERIC_MONITORING_COMMON (p. 1402)

5.207.2.27 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. 1405) 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. 1405) in order to throttle application data flow.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Generic.StrictReliable.LargeData"

5.207.2.28 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. 1404) 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. 1405) in order to throttle application data flow.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Generic.KeepLastReliable.LargeData"

5.207.2.29 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 (~ 100 MB/sec) flow controller.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Generic.StrictReliable.LargeData.FastFlow"

5.207.2.30 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 (~ 25 MB/sec) flow controller.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Generic.StrictReliable.LargeData.MediumFlow"

5.207.2.31 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 (~ 6.25 MB/sec) flow controller.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Generic.StrictReliable.LargeData.SlowFlow"

5.207.2.32 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 (~ 100 MB/sec) flow controller.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Generic.KeepLastReliable.LargeData.FastFlow"

5.207.2.33 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 (~ 25 MB/sec) flow controller.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Generic.KeepLastReliable.LargeData.MediumFlow"

5.207.2.34 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 (~ 6.25 MB/sec) flow controller.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Generic.KeepLastReliable.LargeData.SlowFlow"

5.207.2.35 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. 1404) profile, but persists the samples of a **DDS_DataWriter** (p. 485) as long as the entity exists in order to deliver them to late-joining DataReaders.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Generic.KeepLastReliable.TransientLocal"

5.207.2.36 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. 1404) profile, but persists samples using Persistence Service in order to deliver them to late-joining DataReaders.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Generic.KeepLastReliable.Transient"

5.207.2.37 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. 1404) 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. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Generic.KeepLastReliable.Persistent"

5.207.2.38 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. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Generic.AutoTuning"

5.207.2.39 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. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Generic.MinimalMemoryFootprint"

5.207.2.40 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">
  </domain_participant_qos>
</qos_profile>
```

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400)

String-version: "Generic.Security"

5.207.2.41 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. 1410).

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. 1426).

String-version: "Generic.Monitoring2"

See also

RTI_MONITORING_PERIODIC_TOPIC_NAME (p. 1436)

RTI_MONITORING_EVENT_TOPIC_NAME (p. 1436)

RTI_MONITORING_LOGGING_TOPIC_NAME (p. 1436)

5.207.2.42 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. 1410).

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. 1426).

String-version: "Generic.Monitoring2.WAN"

See also

DDS_PROFILE_GENERIC_MONITORING2 (p. 1410)

5.207.2.43 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. 1415)

String-version: "Pattern.RPC"

5.207.2.44 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. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Pattern.PeriodicData"

5.207.2.45 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. 1412) 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. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Pattern.Streaming"

5.207.2.46 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. 1404) QoS Profile and the following QoS Snippets:

- **DDS_SNIPPET_QOS_POLICY_RELIABILITY_RELIABLE** (p. 1420)
- **DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_LAST_1** (p. 1421)
- **DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_LAST** (p. 1416)

This QoS Snippet also increases the **DDS_HistoryQosPolicy::depth** (p. 1807) to reduce the probability of losing samples.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Pattern.ReliableStreaming"

5.207.2.47 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. 1420)
- **DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_ALL** (p. 1421)
- **DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_ALL** (p. 1415)

Since events and alerts are critical and non-periodic data, it is important to detect situations in which communication between a **DDS_DataWriter** (p. 485) and **DDS_DataReader** (p. 615) is broken. This is why this profile sets the **DDS_LivelinessQosPolicy** (p. 1823). If the **DDS_DataWriter** (p. 485) does not assert its liveliness in a timely manner, the **DDS_DataReader** (p. 615) will report 'loss of liveliness' to the application.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Pattern.Event"

5.207.2.48 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. 1413).

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Pattern.AlarmEvent"

5.207.2.49 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. 1412) 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. 1412) QoS Profile and publish the data at a fixed rate or it could use the **DDS_PROFILE_PATTERN_STATUS** (p. 1414) QoS Profile and only publish the temperature when it changes more than 1 degree.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Pattern.Status"

5.207.2.50 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. 1414).

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Pattern.AlarmStatus"

5.207.2.51 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 **DDS_DataWriter** (p. 485) 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. 1409) because the use case requires delivery to late-joiners.

In Library: **DDS_BUILTIN_QOS_LIB** (p. 1400) and **DDS_BUILTIN_QOS_LIB_EXP** (p. 1404)

String-version: "Pattern.LastValueCache"

5.207.2.52 DDS_BUILTIN_QOS_SNIPPET_LIB

```
const char* const DDS_BUILTIN_QOS_SNIPPET_LIB [extern]
```

A library of QoS Snippets.

String-version: "BuiltinQosSnippetLib"

5.207.2.53 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. 1653)
- **DDS_DataReaderQos** (p. 1611) => **DDS_ResourceLimitsQosPolicy** (p. 1949)
- **DDS_DataReaderProtocolQosPolicy::rtps_reliable_reader** (p. 1599)

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. 1966), which works in combination with the heartbeat rate to determine when the **DDS_DataWriter** (p. 485) considers a **DDS_DataReader** (p. 615) non-responsive. This QoS Snippet configures the **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 1653) parameters so that the **DDS_DataWriter** (p. 485) considers the **DDS_DataReader** (p. 615) to be "inactive" after 500 unresponded heartbeats.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "Optimization.ReliabilityProtocol.Common"

5.207.2.54 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 **DDS_DataWriter** (p. 485) reliable protocol parameters for a reliable **DDS_DataWriter** (p. 485) with **DDS_HistoryQosPolicyKind** (p. 1292) set to **DDS_KEEP_ALL_HISTORY_QOS** (p. 1293).

Modified QoS Parameters:

- **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 1653)
- **DDS_DataReaderQos** (p. 1611) => **DDS_ResourceLimitsQosPolicy** (p. 1949)
- **DDS_DataReaderProtocolQosPolicy::rtps_reliable_reader** (p. 1599)

Note that this QoS Snippet does not configure the **DDS_ReliabilityQosPolicy** (p. 1936) or **DDS_HistoryQosPolicy** (p. 1805) 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. 1415)

String-version: "Optimization.ReliabilityProtocol.KeepAll"

5.207.2.55 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 **DDS_DataWriter** (p. 485) reliable protocol parameters for a reliable **DDS_DataWriter** (p. 485) with **DDS_HistoryQosPolicyKind** (p. 1292) set to **DDS_KEEP_LAST_HISTORY_QOS** (p. 1293).

Modified QoS Parameters:

- **DDS_DataWriterQos** (p. 1666) => **DDS_ResourceLimitsQosPolicy** (p. 1949)
- **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 1653)
- **DDS_DataReaderQos** (p. 1611) => **DDS_ResourceLimitsQosPolicy** (p. 1949)
- **DDS_DataReaderProtocolQosPolicy::rtps_reliable_reader** (p. 1599)

Note that this QoS Snippet does not configure the **DDS_ReliabilityQosPolicy** (p. 1936) or **DDS_HistoryQosPolicy** (p. 1805) 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. 1415)

String-version: "Optimization.ReliabilityProtocol.KeepLast"

5.207.2.56 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 **DDS_DataWriter** (p. 485) reliable protocol parameters for a reliable **DDS_DataWriter** (p. 485) 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. 1653)
- **DDS_DataReaderQos** (p. 1611) => **DDS_ResourceLimitsQosPolicy** (p. 1949)
- **DDS_DataReaderProtocolQosPolicy::rtps_reliable_reader** (p. 1599)

This QoS Snippet sets a fast rate of heartbeats so that errors are detected and repaired more swiftly.

Note that to get the highest throughput you may need to apply additional changes to the final QoS Profile. See the QoS Profile **DDS_PROFILE_GENERIC_STRICT_RELIABLE_HIGH_THROUGHPUT** (p. 1405) for further information.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "Optimization.ReliabilityProtocol.HighRate"

5.207.2.57 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. 1653)
- **DDS_DataReaderQos** (p. 1611) => **DDS_ResourceLimitsQosPolicy** (p. 1949)
- **DDS_DataReaderProtocolQosPolicy::rtps_reliable_reader** (p. 1599)

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. 1405) for further information.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "Optimization.ReliabilityProtocol.LowLatency"

5.207.2.58 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. 1653)
- **DDS_DataReaderQos** (p. 1611) => **DDS_ResourceLimitsQosPolicy** (p. 1949)
- **DDS_DataReaderProtocolQosPolicy::rtps_reliable_reader** (p. 1599)

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. 1407), and derivatives for fully functional Large Data QoS Profiles.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "Optimization.ReliabilityProtocol.LargeData"

5.207.2.59 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. 1666) => **DDS_ResourceLimitsQosPolicy** (p. 1949)
- **DDS_DataWriterQos** (p. 1666) => **DDS_PropertyQosPolicy** (p. 1901) named "dds.data_writer.history.↔memory_manager.*"
- **DDS_DataReaderQos** (p. 1611) => **DDS_ResourceLimitsQosPolicy** (p. 1949)
- **DDS_DataReaderResourceLimitsQosPolicy** (p. 1619)
- **DDS_DataReaderQos** (p. 1611) => **DDS_PropertyQosPolicy** (p. 1901) 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. 1415)

String-version: "Optimization.DataCache.LargeData.DynamicMemAlloc"

5.207.2.60 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. 1726) 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. 1692)
- **DDS_DiscoveryConfigQosPolicy::participant_liveliness_assert_period** (p. 1692)
- **DDS_DiscoveryConfigQosPolicy::max_liveliness_loss_detection_period** (p. 1694)
- **DDS_DiscoveryConfigQosPolicy::initial_participant_announcements** (p. 1694)
- **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 1698)
- **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 1700)

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "Optimization.Discovery.Common"

5.207.2.61 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. 1726) => **DDS_PropertyQosPolicy** (p. 1901) named "dds.participant.inter_↔ participant.*"
- **DDS_DomainParticipantQos** (p. 1726) => **DDS_PropertyQosPolicy** (p. 1901) named "dds.sys_info.*"

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "Optimization.Discovery.Participant.Compact"

5.207.2.62 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. 1698)
- **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 1700)

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "Optimization.Discovery.Endpoint.Fast"

5.207.2.63 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 shm and udpv4 use.

This is useful when using Large Data

Modified QoS Parameters:

- **DDS_ReceiverPoolQosPolicy** (p. 1934)
- **DDS_TransportBuiltinQosPolicy** (p. 2051)

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "Optimization.Transport.LargeBuffers"

5.207.2.64 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 **DDS_DataWriter** (p. 485) writes faster than the DataReaders can accommodate.

Modified QoS Parameters:

- **DDS_DataWriterQos** (p. 1666) => **DDS_ReliabilityQosPolicy** (p. 1936)
- **DDS_DataReaderQos** (p. 1611) => **DDS_ReliabilityQosPolicy** (p. 1936)

Note that by itself enabling reliability does not ensure that every sample written is delivered to the DataReaders. This is because the **DDS_DataWriter** (p. 485) and/or **DDS_DataReader** (p. 615) can be configured to override samples in its cache based on the configuration of the **DDS_HistoryQosPolicy** (p. 1805) QoS policy.

To ensure delivery of every sample (at the expense of potentially blocking the **DDS_DataWriter** (p. 485)), use the QoS Profile **DDS_PROFILE_GENERIC_STRICT_RELIABLE** (p. 1404) or one of the derived QoS Profiles.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "QosPolicy.Reliability.Reliable"

5.207.2.65 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. 1666) => **DDS_ReliabilityQosPolicy** (p. 1936)
- **DDS_DataReaderQos** (p. 1611) => **DDS_ReliabilityQosPolicy** (p. 1936)

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. 1415)

String-version: "QosPolicy.Reliability.BestEffort"

5.207.2.66 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. 1666) => **DDS_HistoryQosPolicy** (p. 1805)
- **DDS_DataReaderQos** (p. 1611) => **DDS_HistoryQosPolicy** (p. 1805)

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "QosPolicy.History.KeepLast_1"

5.207.2.67 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. 1666) => **DDS_HistoryQosPolicy** (p. 1805)
- **DDS_DataReaderQos** (p. 1611) => **DDS_HistoryQosPolicy** (p. 1805)

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "QosPolicy.History.KeepAll"

5.207.2.68 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. 1920)

Asynchronous Publish mode decouples the application thread that calls the **DDS_DataWriter** (p.485) "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. 1415)

String-version: "QosPolicy.PublishMode.Asynchronous"

5.207.2.69 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. 1666) => **DDS_DurabilityQosPolicy** (p. 1759)
- **DDS_DataReaderQos** (p. 1611) => **DDS_DurabilityQosPolicy** (p. 1759)

DataWriters will store and send previously published DDS samples for delivery to newly discovered DataReaders as long as the **DDS_DataWriter** (p.485) still exists. For this setting to be effective, you must also set the **DDS_ReliabilityQosPolicyKind** (p. 1322) to **DDS_RELIABLE_RELIABILITY_QOS** (p. 1323) (not Best Effort). Which particular DDS samples are kept depends on other QoS settings such as **DDS_HistoryQosPolicy** (p. 1805) and **DDS_ResourceLimitsQosPolicy** (p. 1949).

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "QosPolicy.Durability.TransientLocal"

5.207.2.70 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. 1666) => **DDS_DurabilityQosPolicy** (p. 1759)
- **DDS_DataReaderQos** (p. 1611) => **DDS_DurabilityQosPolicy** (p. 1759)

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. 1805) and **DDS_ResourceLimitsQosPolicy** (p. 1949) of the Persistence Service DataWriters. These QosPolicies can be configured in the Persistence Service configuration file or through the **DDS_DurabilityQosPolicyKind** (p. 1287) of the DataWriters configured with **DDS_TRANSIENT_DURABILITY_QOS** (p. 1288).

You need a Persistence Service instance running to use this behavior.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "QosPolicy.Durability.Transient"

5.207.2.71 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. 1666) => **DDS_DurabilityQosPolicy** (p. 1759)
- **DDS_DataReaderQos** (p. 1611) => **DDS_DurabilityQosPolicy** (p. 1759)

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. 1805) and **DDS_ResourceLimitsQosPolicy** (p. 1949) in the Persistence Service DataWriters. These QosPolicies can be configured in the Persistence Service configuration file or through the **DDS_DurabilityQosPolicyKind** (p. 1287) of the DataWriters configured with **DDS_PERSISTENT_DURABILITY_QOS** (p. 1288).

You need a Persistence Service instance running to use this behavior.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "QosPolicy.Durability.Persistent"

5.207.2.72 DDS_SNIPPET_QOS_POLICY_BATCHING_ENABLE

```
const char* const DDS_SNIPPET_QOS_POLICY_BATCHING_ENABLE [extern]
```

QoS Snippet that sets BATCH QoS Policy to true.

Modified QoS Parameters:

- **DDS_BatchQosPolicy** (p. 1552)

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. 1415)

String-version: "QoSPolicy.Batching.Enable"

5.207.2.73 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 **DDS_FlowController** (p. 559) and configures the **DDS_DataWriterQos** (p. 1666) with it.

This is a **DDS_FlowController** (p. 559) of 838 Mbps (~ 100 MB/sec)

Modified QoS Parameters:

- **DDS_DomainParticipantQos** (p. 1726) => **DDS_PropertyQosPolicy** (p. 1901) named "dds.flow_controller.↔ token_bucket.*"
- **DDS_PublishModeQosPolicy** (p. 1920)

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "Feature.FlowController.838Mbps"

5.207.2.74 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 **DDS_FlowController** (p. 559) and configures the **DDS_DataWriterQos** (p. 1666) with it.

This is a **DDS_FlowController** (p. 559) of 209 Mbps (~ 25 MB/sec)

Modified QoS Parameters:

- **DDS_DomainParticipantQos** (p. 1726) => **DDS_PropertyQosPolicy** (p. 1901) named "dds.flow_controller.↔ token_bucket.*"
- **DDS_PublishModeQosPolicy** (p. 1920)

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "Feature.FlowController.209Mbps"

5.207.2.75 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 **DDS_FlowController** (p. 559) and configures the **DDS_DataWriterQos** (p. 1666) with it.

This is a **DDS_FlowController** (p. 559) of 52 Mbps (~ 6.25 MB/sec)

Modified QoS Parameters:

- **DDS_DomainParticipantQos** (p. 1726) => **DDS_PropertyQosPolicy** (p. 1901) named "dds.flow_controller.token_bucket.*"
- **DDS_PublishModeQosPolicy** (p. 1920)

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "Feature.FlowController.52Mbps"

5.207.2.76 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. 1726) properties to enable auto_throttle and turbo_mode to true.

Modified QoS Parameters:

- **DDS_DomainParticipantQos** (p. 1726) => **DDS_PropertyQosPolicy** (p. 1901) named "dds.domain_participant.auto_throttle"
- **DDS_DataWriterQos** (p. 1666) => **DDS_PropertyQosPolicy** (p. 1901) named "dds.data_writer.auto_throttle.enable"
- **DDS_DataWriterQos** (p. 1666) => **DDS_PropertyQosPolicy** (p. 1901) named "dds.data_writer.enable_turbo_mode"

The domain_participant.auto_throttle configures the **DDS_DomainParticipant** (p. 77) to gather internal measurements (during **DDS_DomainParticipant** (p. 77) creation) that are required for the Auto Throttle feature. This allows DataWriters belonging to this **DDS_DomainParticipant** (p. 77) to use the Auto Throttle feature.

The turbo_mode adjusts the batch max_data_bytes based on how frequently the **DDS_DataWriter** (p. 485) writes data.

Data_writer.auto_throttle enables automatic throttling in the **DDS_DataWriter** (p. 485) 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. 1415)

String-version: "Feature.AutoTuning.Enable"

5.207.2.77 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 **DDS_DomainParticipant** (p. 77). 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. 1415)

String-version: "Feature.Monitoring.Enable"

5.207.2.78 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 DomainParticipantFactory. 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. 1415)

String-version: "Feature.Monitoring2.Enable"

5.207.2.79 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 **DDS_DomainParticipant** (p. 77). 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. 1415)

String-version: "Feature.Security.Enable"

5.207.2.80 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 **DDS_TopicQuery** (p. 707) feature, apply this QoS Snippet to the QoS Profile used to create your **DDS_DataWriter** (p. 485). 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. 1415)

String-version: "Feature.TopicQuery.Enable"

5.207.2.81 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 **DDS_Entity** (p. 1360). These new values will overwrite the current invalid values.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "Transport.TCP.LAN.Client"

5.207.2.82 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 **DDS_Entity** (p. 1360). These new values will overwrite the current invalid values.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "Transport.TCP.WAN.Symmetric.Client"

5.207.2.83 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 **DDS_Entity** (p. 1360). These new values will overwrite the current invalid values.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "Transport.TCP.WAN.Asymmetric.Server"

5.207.2.84 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. 1428)). This modification should be done in the QoS Profile that will be used to create the **DDS_Entity** (p. 1360). This new value will overwrite the current invalid value.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "Transport.TCP.WAN.Asymmetric.Client"

5.207.2.85 DDS_SNIPPET_TRANSPORT_UDP_AVOID_IP_FRAGMENTATION

```
const char* const DDS_SNIPPET_TRANSPORT_UDP_AVOID_IP_FRAGMENTATION [extern]
```

QoS Snippet that configures RTI Connex 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 Connex applications using the built-in UDP transports. Instead, Connex 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 Connex 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. 1415)

String-version: "Transport.UDP.AvoidIPFragmentation"

5.207.2.86 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. 1415)

String-version: "Transport.UDP.WAN"

5.207.2.87 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 Connex Micro 2.4.3.

QoS Snippet that sets the **DDS_DataReaderQos** (p. 1611) and **DDS_DataWriterQos** (p. 1666) **DDS_LivelinessQos**↔ **PolicyKind** (p. 1296) to **DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS** (p. 1297) It also disables the built-in shared memory transport.

Modified QoS Parameters:

- **DDS_DataReaderQos** (p. 1611) => **DDS_LivelinessQosPolicy** (p. 1823)
- **DDS_DataWriterQos** (p. 1666) => **DDS_LivelinessQosPolicy** (p. 1823)
- **DDS_TransportBuiltinQosPolicy** (p. 2051)

RTI Connex Micro versions 2.4.3 and earlier only supported **DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS** (p. 1297) **DDS_LivelinessQosPolicyKind** (p. 1296). In order to be compatible with RTI Connex Micro 2.4.3, the **DDS_Data**↔ **Reader** (p. 615) and **DDS_DataWriter** (p. 485) must have their **DDS_LivelinessQosPolicyKind** (p. 1296) changed to this value because the default kind in RTI Connex is **DDS_AUTOMATIC_LIVELINESS_QOS** (p. 1297).

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "Compatibility.ConnexMicro.Version243"

5.207.2.88 DDS_SNIPPET_COMPATIBILITY_OTHER_DDS_VENDORS_ENABLE

```
const char* const DDS_SNIPPET_COMPATIBILITY_OTHER_DDS_VENDORS_ENABLE [extern]
```

QoS Snippet that configures RTI Connex to interoperate with other DDS vendors.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "Compatibility.OtherDDSVendor.Enable"

5.207.2.89 DDS_SNIPPET_5_1_0_TRANSPORT_ENABLE

```
const char* const DDS_SNIPPET_5_1_0_TRANSPORT_ENABLE [extern]
```

QoS Snippet that configures RTI Connex to interoperate with RTI Connex 5.1.0 and below for UDPv6 and SHMEM transports.

In Library: **DDS_BUILTIN_QOS_SNIPPET_LIB** (p. 1415)

String-version: "Compatibility.510Transport.Enable"

5.208 DomainParticipantConfigParams

<<*extension*>> (p. 1012) **DDS_DomainParticipantConfigParams_t** (p. 1719)

Data Structures

- struct **DDS_DomainParticipantConfigParams_t**
 <<*extension*>> (p. 1012) Input parameters 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. 1012) Initializer for new **DDS_DomainParticipantConfigParams_t** (p. 1719).

Variables

- const int **DDS_DOMAIN_ID_USE_XML_CONFIG**
 <<*extension*>> (p. 1012) Special value to be used with **DDS_DomainParticipantConfigParams_t** (p. 1719) 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. 1012) Special value to be used with **DDS_DomainParticipantConfigParams_t** (p. 1719) to indicate that a participant created is with an autogenerated entity name.
- const char * **DDS_QOS_ELEMENT_NAME_USE_XML_CONFIG**
 <<*extension*>> (p. 1012) Special value to be used with **DDS_DomainParticipantConfigParams_t** (p. 1719) to indicate that entities are created from the QoS profile specified in the participant configuration.

5.208.1 Detailed Description

<<*extension*>> (p. 1012) **DDS_DomainParticipantConfigParams_t** (p. 1719)

5.208.2 Macro Definition Documentation

5.208.2.1 DDS_DomainParticipantConfigParams_t_INITIALIZER

```
#define DDS_DomainParticipantConfigParams_t_INITIALIZER
```

<<*extension*>> (p. 1012) Initializer for new **DDS_DomainParticipantConfigParams_t** (p. 1719).

No memory is allocated. New **DDS_DomainParticipantConfigParams_t** (p. 1719) instances stored in the stack should be initialized with this value before they are passed to any functions. .

5.208.3 Variable Documentation

5.208.3.1 DDS_DOMAIN_ID_USE_XML_CONFIG

```
const int DDS_DOMAIN_ID_USE_XML_CONFIG [extern]
```

<<*extension*>> (p. 1012) Special value to be used with **DDS_DomainParticipantConfigParams_t** (p. 1719) 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.

5.208.3.2 DDS_ENTITY_NAME_USE_XML_CONFIG

```
const char* DDS_ENTITY_NAME_USE_XML_CONFIG [extern]
```

<<*extension*>> (p. 1012) Special value to be used with **DDS_DomainParticipantConfigParams_t** (p. 1719) 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.

5.208.3.3 DDS_QOS_ELEMENT_NAME_USE_XML_CONFIG

```
const char* DDS_QOS_ELEMENT_NAME_USE_XML_CONFIG [extern]
```

<<*extension*>> (p. 1012) Special value to be used with **DDS_DomainParticipantConfigParams_t** (p. 1719) 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.

5.209 User-managed Threads

User-managed thread infrastructure.

Data Structures

- struct **DDS_ThreadFactory**

<<*extension*>> (p. 1012) <<*interface*>> (p. 1013) Interface for providing the threads needed by the middleware. It consists of operations to create and delete threads. The interface depends on the **DDS_ThreadFactory_OnSpawned**↔**Function** (p. 1433) that specifies the operation to run in the new thread.

Macros

- #define **DDS_ThreadFactory_INITIALIZER**

<<*extension*>> (p. 1012) Initializer for new **DDS_ThreadFactory** (p. 2027).

Typedefs

- typedef void (*)(**DDS_ThreadFactory_OnSpawnedFunction**) (void *thread_param)
*Prototype of the function that must be called on a new thread created by a **DDS_ThreadFactory** (p. 2027).*
- typedef void (*)(**DDS_ThreadFactory_CreateThreadCallback**) (void *factory_data, const char *thread_name, const struct **DDS_ThreadSettings_t** *settings, **DDS_ThreadFactory_OnSpawnedFunction** on_spawned, void *threadParam)
*Prototype of a **DDS_ThreadFactory** (p. 2027) create_thread function.*
- typedef void (*)(**DDS_ThreadFactory_DeleteThreadCallback**) (void *factory_data, void *thread)
*Prototype of a **DDS_ThreadFactory** (p. 2027) delete_thread function.*

5.209.1 Detailed Description

User-managed thread infrastructure.

Core feature that allows users to provide the threads to RTI Connext.

"::DDS_ThreadFactory"

The model follows the abstract factory pattern. A **DDS_ThreadFactory** (p. 2027) instance can be set in the **DDS_**↔**DomainParticipantFactory** (p. 32) so that **DDS_DomainParticipant** (p. 77) will use it for thread creation and deletion.

5.209.2 Macro Definition Documentation

5.209.2.1 DDS_ThreadFactory_INITIALIZER

```
#define DDS_ThreadFactory_INITIALIZER
```

<<**extension**>> (p. 1012) Initializer for new **DDS_ThreadFactory** (p. 2027).

No memory is allocated. New **DDS_ThreadFactory** (p. 2027) instances stored in the stack should be initialized with this value before they are passed to any functions.

5.209.3 Typedef Documentation

5.209.3.1 DDS_ThreadFactory_OnSpawnedFunction

```
typedef void *(* DDS_ThreadFactory_OnSpawnedFunction) (void *thread_param)
```

Prototype of the function that must be called on a new thread created by a **DDS_ThreadFactory** (p. 2027).

Parameters

<i>thread_param</i>	<< in >> (p. 1013) Single argument that this operation receives. It is the data this operation needs to perform.

5.209.3.2 DDS_ThreadFactory_CreateThreadCallback

```
typedef void *(* DDS_ThreadFactory_CreateThreadCallback) (void *factory_data, const char *thread_name, const struct DDS_ThreadSettings_t *settings, DDS_ThreadFactory_OnSpawnedFunction on_spawned, void *threadParam)
```

Prototype of a **DDS_ThreadFactory** (p. 2027) create_thread function.

Parameters

<i>factory_data</i>	<< in >> (p. 1013) Data associated with the factory when the factory is set.
<i>thread_name</i>	<< in >> (p. 1013) Name of the thread given by the middleware.
<i>settings</i>	<< in >> (p. 1013) Attributes of the new thread (i.e., priority, stack size, etc.). These attributes are specified in the QoS settings corresponding to each kind of thread (i.e., database, event, etc).

Parameters

<i>on_spawned</i>	<< <i>in</i> >> (p. 1013) Function provided by the middleware to be called in the newly created thread.
<i>threadParam</i>	<< <i>in</i> >> (p. 1013) Parameters provided by the middleware to be passed in the call to <i>on_spawned</i> .

Returns

An opaque pointer to the newly created thread. The same pointer will be provided to the **DDS_ThreadFactory**↔
::delete_thread (p. 2028) operation when deletion is requested.

5.209.3.3 DDS_ThreadFactory_DeleteThreadCallback

```
typedef void(* DDS_ThreadFactory_DeleteThreadCallback) (void *factory_data, void *thread)
```

Prototype of a **DDS_ThreadFactory** (p. 2027) *delete_thread* function.

Parameters

<i>factory_data</i>	<< <i>in</i> >> (p. 1013) Data associated with the factory when the factory is set.
<i>thread</i>	<< <i>in</i> >> (p. 1013) Thread to be deleted. The thread was previously created by the same DDS_ThreadFactory (p. 2027).

5.210 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.

5.210.1 Detailed Description

RTI Monitoring Library 2.0 public API.

RTI Monitoring Library 2.0 is one component of the RTI Connex Observability Framework which allows collecting and distributing telemetry data (metrics and logs) associated with the observable resources created by an RTI Connex application.

In this release, the only Observable resources are the following entities: **DDS_DataWriter** (p. 485), **DDS_DataReader** (p. 615), **DDS_Publisher** (p. 443), **DDS_Subscriber** (p. 572), **DDS_DomainParticipant** (p. 77), **DDS_Topic** (p. 175) and Application (a process running RTI Connex).

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 (rtmonitoring2), and applications can use it in three different modes:

- **Dynamically loaded:** This is the default mode, and it requires that the rtmonitoring2 shared library is in the library search path.
- **Dynamic linking:** The application is linked with the rtmonitoring2 shared library.
- **Static linking:** The application is linked with the rtmonitoring2 static library.

The last two modes require calling the API **RTI_Monitoring_initialize** (p. 1435) in your application before any other RTI Connex 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. 1853) QoS policy on the **DDS_DomainParticipantFactory** (p. 32). This QoS policy can be configured programmatically or via XML.

5.210.2 Function Documentation

5.210.2.1 RTI_Monitoring_initialize()

```
void RTI_Monitoring_initialize (  
    void )
```

Initializes Monitoring Library 2.0.

This function must be called before calling any other RTI Connex API if the application links with RTI Monitoring Library 2.0 dynamically or statically.

5.210.3 Variable Documentation

5.210.3.1 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.

5.210.3.2 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.

5.210.3.3 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.

5.211 Version

Retrieve information for the RTI Connext product, the core library, and the C, C++ or Java libraries.

Data Structures

- struct **NDDS_Config_LibraryVersion_t**
The version of a single library shipped as part of an RTI Connext distribution.
- struct **NDDS_Config_Version_t**
<<*interface*>> (p. 1013) *The version of an RTI Connext distribution.*

Functions

- const struct **DDS_ProductVersion_t** * **NDDS_Config_Version_get_product_version** (void)
Get the RTI Connext product version.
- const struct **NDDS_Config_LibraryVersion_t** * **NDDS_Config_Version_get_api_version** (void)
Get the version of the C API library.
- const struct **NDDS_Config_LibraryVersion_t** * **NDDS_Config_Version_get_core_version** (void)
Get the version of the core library.
- const char * **NDDS_Config_Version_to_string** (void)
Get this version in string form.

5.211.1 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.

5.211.2 Function Documentation

5.211.2.1 NDDS_Config_Version_get_product_version()

```
const struct DDS_ProductVersion_t * NDDS_Config_Version_get_product_version (
    void )
```

Get the RTI Connext product version.

5.211.2.2 NDDS_Config_Version_get_api_version()

```
const struct NDDS_Config_LibraryVersion_t * NDDS_Config_Version_get_api_version (
    void )
```

Get the version of the C API library.

5.211.2.3 NDDS_Config_Version_get_core_version()

```
const struct NDDS_Config_LibraryVersion_t * NDDS_Config_Version_get_core_version (
    void )
```

Get the version of the core library.

5.211.2.4 NDDS_Config_Version_to_string()

```
const char * NDDS_Config_Version_to_string (
    void )
```

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 is internal to this **NDDS_Config_Version_t** (p. 2119). The caller should not modify it.

5.212 Logging

Configure how much debugging information is reported during runtime and where it is logged.

Modules

- **Activity Context**

Add contextual information to log messages.

Data Structures

- struct **NDDS_Config_Logger**

<<interface>> (p. 1013) The singleton type used to configure RTI Connex logging.

- struct **NDDS_Config_LogMessage**

A log message's input parameter to a logging device, including the text and additional information.

- struct **NDDS_Config_LoggerDevice**

<<interface>> (p. 1013) Logging device interface. Use for user-defined logging devices.

Macros

- `#define NDDS_Config_LoggerDevice_INITIALIZER`

Initializer for new Logger Device instances.

Typedefs

- `typedef void(* NDDS_Config_LoggerDeviceWriteFnc) (struct NDDS_Config_LoggerDevice *device, const struct NDDS_Config_LogMessage *message)`

*Prototype of a **NDDS_Config_LoggerDevice** (p. 2116) write function.*

- `typedef void(* NDDS_Config_LoggerDeviceCloseFnc) (struct NDDS_Config_LoggerDevice *device)`

*Prototype of a **NDDS_Config_LoggerDevice** (p. 2116) close function.*

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 verbositys at which RTI Connex 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 Connex 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 Connex 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 Connex 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 verbosity at which RTI Connex diagnostic information is logged. These Syslog verbosity are mapped to **NDDS_Config_LogVerbosity** (p. 1443). See *Syslog Level and Verbosity Mapping*, in the *Core Libraries User's Manual*, for more information.*

Functions

- **NDDS_Config_Logger * NDDS_Config_Logger_get_instance** (void)
Get the singleton instance of this type.
- **NDDS_Config_Logger * NDDS_Config_Logger_get_instancel** (void)
Get the singleton instance of this type.
- **NDDS_Config_LogVerbosity NDDS_Config_Logger_get_verbosity** (const **NDDS_Config_Logger** *self)
Get the verbosity at which RTI Connex is currently logging diagnostic information.
- **NDDS_Config_LogVerbosity NDDS_Config_Logger_get_verbosity_by_category** (const **NDDS_Config_Logger** *self, **NDDS_Config_LogCategory** category)
Get the verbosity at which RTI Connex is currently logging diagnostic information in the given category.
- void **NDDS_Config_Logger_set_verbosity** (**NDDS_Config_Logger** *self, **NDDS_Config_LogVerbosity** verbosity)
Set the verbosity at which RTI Connex will log diagnostic information.
- void **NDDS_Config_Logger_set_verbosity_by_category** (**NDDS_Config_Logger** *self, **NDDS_Config_LogCategory** category, **NDDS_Config_LogVerbosity** verbosity)
Set the verbosity at which RTI Connex will log diagnostic information in the given category.
- FILE * **NDDS_Config_Logger_get_output_file** (**NDDS_Config_Logger** *self)
Get the file to which the logged output is redirected.
- **DDS_Boolean NDDS_Config_Logger_set_output_file** (**NDDS_Config_Logger** *self, FILE *out)
Set the file to which the logged output is redirected.
- **DDS_Boolean NDDS_Config_Logger_set_output_file_name** (**NDDS_Config_Logger** *self, const char *file_name)
Set the name of the file to which the logged output is redirected.
- **DDS_Boolean NDDS_Config_Logger_set_output_file_set** (**NDDS_Config_Logger** *self, const char *file_prefix, const char *file_suffix, int max_capacity, int max_files)

Configure a set of files to redirect the logged output.

- **NDDS_Config_LogPrintFormat** **NDDS_Config_Logger_get_print_format** (const **NDDS_Config_Logger** *self)

Get the current message format for the log level **NDDS_CONFIG_LOG_LEVEL_ERROR** (p. 1444).

- **NDDS_Config_LogPrintFormat** **NDDS_Config_Logger_get_print_format_by_log_level** (const **NDDS_Config_Logger** *self, **NDDS_Config_LogLevel** log_level)

Get the current message format, by log level, that RTI Connex is using to log diagnostic information.

- **DDS_Boolean** **NDDS_Config_Logger_set_print_format** (**NDDS_Config_Logger** *self, **NDDS_Config_LogPrintFormat** print_format)

Set the message format that RTI Connex will use to log diagnostic information for all the log levels, except for **NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR** (p. 1444). When the **Activity Context** (p. 1458) is printed, the user can select the information that will be part of the **Activity Context** (p. 1458) by using the API **NDDS_Config_ActivityContext_set_attribute_mask** (p. 1462).

- **DDS_Boolean** **NDDS_Config_Logger_set_print_format_by_log_level** (**NDDS_Config_Logger** *self, **NDDS_Config_LogPrintFormat** print_format, **NDDS_Config_LogLevel** log_level)

Set the message format that RTI Connex will use to log diagnostic information for all the log levels, except for **NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR** (p. 1444). When the **Activity Context** (p. 1458) is printed, the user can select the information that will be part of the **Activity Context** (p. 1458) by using the API **NDDS_Config_ActivityContext_set_attribute_mask** (p. 1462).

- void **NDDS_Config_Logger_emergency** (**NDDS_Config_Logger** *self, const char *msg)

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_EMERGENCY** (p. 1444) and **NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1446). **Note:** **DDS_DomainParticipantFactory_finalize_instance** (p. 38) needs to be called to finalize the Logger singleton. .

- void **NDDS_Config_Logger_alert** (**NDDS_Config_Logger** *self, const char *msg)

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_ALERT** (p. 1444) and **NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1446). **Note:** **DDS_DomainParticipantFactory_finalize_instance** (p. 38) needs to be called to finalize the Logger singleton. .

- void **NDDS_Config_Logger_critical** (**NDDS_Config_Logger** *self, const char *msg)

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_CRITICAL** (p. 1445) and **NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1446). **Note:** **DDS_DomainParticipantFactory_finalize_instance** (p. 38) needs to be called to finalize the Logger singleton. .

- void **NDDS_Config_Logger_error** (**NDDS_Config_Logger** *self, const char *msg)

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_ERROR** (p. 1445) and **NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1446). **Note:** **DDS_DomainParticipantFactory_finalize_instance** (p. 38) needs to be called to finalize the Logger singleton. .

- void **NDDS_Config_Logger_warning** (**NDDS_Config_Logger** *self, const char *msg)

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_WARNING** (p. 1445) and **NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1446). **Note:** **DDS_DomainParticipantFactory_finalize_instance** (p. 38) needs to be called to finalize the Logger singleton. .

- void **NDDS_Config_Logger_notice** (**NDDS_Config_Logger** *self, const char *msg)

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_NOTICE** (p. 1445) and **NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1446). **Note:** **DDS_DomainParticipantFactory_finalize_instance** (p. 38) needs to be called to finalize the Logger singleton. .

- void **NDDS_Config_Logger_informational** (**NDDS_Config_Logger** *self, const char *msg)

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_INFORMATIONAL** (p. 1445) and **NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1446). **Note:** **DDS_DomainParticipantFactory_finalize_instance** (p. 38) needs to be called to finalize the Logger singleton. .

- void **NDDS_Config_Logger_debug** (**NDDS_Config_Logger** *self, const char *msg)

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_DEBUG** (p. 1445) and **NDDS_CONFIG_LOG_CATEGORY_USER** (p. 1446). **Note:** **DDS_DomainParticipantFactory_finalize_instance** (p. 38) needs to be called to finalize the Logger singleton. .

- struct **NDDS_Config_LoggerDevice** * **NDDS_Config_Logger_get_output_device** (**NDDS_Config_Logger** *self)

Return the user device registered with the logger.

- **DDS_Boolean** **NDDS_Config_Logger_set_output_device** (**NDDS_Config_Logger** *self, struct **NDDS_Config_LoggerDevice** *device)

*Register a **NDDS_Config_LoggerDevice** (p. 2116).*

5.212.1 Detailed Description

Configure how much debugging information is reported during runtime and where it is logged.

5.212.2 Macro Definition Documentation

5.212.2.1 NDDS_Config_LoggerDevice_INITIALIZER

```
#define NDDS_Config_LoggerDevice_INITIALIZER
```

Initializer for new Logger Device instances.

5.212.3 Typedef Documentation

5.212.3.1 NDDS_Config_LoggerDeviceWriteFnc

```
typedef void(* NDDS_Config_LoggerDeviceWriteFnc) (struct NDDS_Config_LoggerDevice *device, const  
struct NDDS_Config_LogMessage *message)
```

Prototype of a **NDDS_Config_LoggerDevice** (p.2116) write function.

Write a log message to the input device.

Note: It is not safe to make any calls to the RTI Connex core library, including calls to **DDS_DomainParticipant_get_↵**
_current_time (p. 138), from any of the logging device operations.

Parameters

<i>device</i>	<< <i>in</i> >> (p. 1013) Logging device.
<i>message</i>	<< <i>in</i> >> (p. 1013) Message to log.

5.212.3.2 NDDS_Config_LoggerDeviceCloseFnc

```
typedef void(* NDDS_Config_LoggerDeviceCloseFnc) (struct NDDS_Config_LoggerDevice *device)
```

Prototype of a **NDDS_Config_LoggerDevice** (p. 2116) close function.

Close the input device.

Note: It is not safe to make any calls to the RTI Connex core library, including calls to **DDS_DomainParticipant_get_↔_current_time** (p. 138), from any of the logging device operations.

Parameters

<i>device</i>	<< <i>in</i> >> (p. 1013) Logging device.

5.212.4 Enumeration Type Documentation

5.212.4.1 NDDS_Config_LogVerbosity

```
enum NDDS_Config_LogVerbosity
```

The verbositys at which RTI Connex 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 Connex. 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 Connex 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 Connex objects will be logged.
NDDS_CONFIG_LOG_VERBOSITY_STATUS_↔ REMOTE	Errors, warnings, and verbose information about the lifecycles of remote RTI Connex objects will be logged.
NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL	Errors, warnings, verbose information about the lifecycles of local and remote RTI Connex objects, and periodic information about RTI Connex threads will be logged.

5.212.4.2 NDDS_Config_LogLevel

enum **NDDS_Config_LogLevel**

Level category assigned to RTI Connex 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 Connex. 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 Connex. 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 Connex 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 Connex objects will be logged.
NDDS_CONFIG_LOG_LEVEL_STATUS_REMOTE	The message contains information about the lifecycles of remote RTI Connex objects will be logged.
NDDS_CONFIG_LOG_LEVEL_DEBUG	The message contains debug information that might be relevant to your application.

5.212.4.3 NDDS_Config_SyslogLevel

enum **NDDS_Config_SyslogLevel**

Syslog level category assigned to RTI Connex 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. 1444).
NDDS_CONFIG_SYSLOG_LEVEL_ALERT	Should be corrected immediately. Maps to NDDS_CONFIG_LOG_LEVEL_ERROR (p. 1444).

Enumerator

NDDS_CONFIG_SYSLOG_LEVEL_CRITICAL	Critical conditions. Maps to NDDS_CONFIG_LOG_LEVEL_ERROR (p. 1444).
NDDS_CONFIG_SYSLOG_LEVEL_ERROR	Error conditions. Maps to NDDS_CONFIG_LOG_LEVEL_ERROR (p. 1444).
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. 1444).
NDDS_CONFIG_SYSLOG_LEVEL_NOTICE	Events that are unusual, but not error conditions. Maps to NDDS_CONFIG_LOG_LEVEL_STATUS_LOCAL (p. 1444).
NDDS_CONFIG_SYSLOG_LEVEL_INFORMATIONAL	Normal operational messages that require no action. Maps to NDDS_CONFIG_LOG_LEVEL_STATUS_LOCAL (p. 1444).
NDDS_CONFIG_SYSLOG_LEVEL_DEBUG	Information useful to developers for debugging the application. Maps to NDDS_CONFIG_LOG_LEVEL_DEBUG (p. 1444).

5.212.4.4 NDDS_Config_LogCategory

```
enum NDDS_Config_LogCategory
```

Categories of logged messages.

The **NDDS_Config_Logger_get_verbosity_by_category** (p. 1449) and **NDDS_Config_Logger_set_verbosity_by_↔_category** (p. 1450) can be used to specify different verbosity levels for different categories of messages.

Enumerator

NDDS_CONFIG_LOG_CATEGORY_PLATFORM	Log messages pertaining to the underlying platform (hardware and OS) on which RTI Connex 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 Connex 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 Connex (such as function argument validation) are in this category.

Enumerator

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 Connex 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: NDDS_Config_Logger_emergency (p. 1452), NDDS_Config_Logger_alert (p. 1453), NDDS_Config_Logger_critical (p. 1453), NDDS_Config_Logger_error (p. 1454), NDDS_Config_Logger_warning (p. 1454), NDDS_Config_Logger_notice (p. 1455), NDDS_Config_Logger_informational (p. 1455), NDDS_Config_Logger_debug (p. 1456).
NDDS_CONFIG_LOG_CATEGORY_ALL	Log messages pertaining to all categories in RTI Connex.

5.212.4.5 NDDS_Config_LogPrintFormat

enum **NDDS_Config_LogPrintFormat**

The format used to output RTI Connex diagnostic information.

Enumerator

NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT	(default) Print message, method name, log level, Activity Context (p. 1458) (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. 1458), logging category, and timestamp.
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.

5.212.4.6 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 Connex 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 NDDS_Config_Logger (p. 2116) (e.g, NDDS_Config_Logger_emergency (p. 1452)). 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 Connex Core Libraries. Numerical code: 23

5.212.4.7 NDDS_Config_SyslogVerbosity

enum **NDDS_Config_SyslogVerbosity**

The Syslog verbosity at which RTI Connex diagnostic information is logged. These Syslog verbosity are mapped to **NDDS_Config_LogVerbosity** (p. 1443). 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. 1443).
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. 1443) when interpreted by RTI Connex. (That's why this level actually logs more than just emergency messages.)

Enumerator

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. 1443) when interpreted by RTI Connex. (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. 1443) when interpreted by RTI Connex. (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. 1443) when interpreted by RTI Connex.
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. 1443) when interpreted by RTI Connex.
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. 1443) when interpreted by RTI Connex. (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. 1443) when interpreted by RTI Connex.
NDDS_CONFIG_SYSLOG_VERBOSITY_DEBUG	All messages will be logged. This Syslog verbosity level is translated to NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL (p. 1443) when interpreted by RTI Connex.

5.212.5 Function Documentation

5.212.5.1 NDDS_Config_Logger_get_instance()

```
NDDS_Config_Logger * NDDS_Config_Logger_get_instance (
    void )
```

Get the singleton instance of this type.

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this function while another thread may be simultaneously calling **DDS_DomainParticipantFactory_get_instance** (p.38), **DDS_DomainParticipantFactory_finalize_instance** (p.38), **DDS_TypeCodeFactory_get_instance** (p.289), **DDS_GuardCondition_new** (p.1376), **DDS_WaitSet_new** (p.1380), **DDS_WaitSet_new_ex** (p.1380), **DDS_GuardCondition_delete** (p.1376), **DDS_WaitSet_delete** (p.1381), **NDDS_Config_Logger_get_instance** (p.1448), **NDDS_Config_Logger_get_output_device** (p.1456), **NDDS_Config_Logger_set_output_device** (p.1457), **DDS_TypeCodeFactory_finalize_instance** (p.289), **NDDS_Utility_enable_network_capture**, or **NDDS_Utility_disable_network_capture**.

5.212.5.2 NDDS_Config_Logger_get_instance()

```
NDDS_Config_Logger * NDDS_Config_Logger_get_instanceI (
    void )
```

Get the singleton instance of this type.

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this function while another thread may be simultaneously calling **DDS_DomainParticipantFactory_get_instance** (p.38), **DDS_DomainParticipantFactory_finalize_instance** (p.38), **DDS_TypeCodeFactory_get_instance** (p.289), **DDS_GuardCondition_new** (p.1376), **DDS_WaitSet_new** (p.1380), **DDS_WaitSet_new_ex** (p.1380), **DDS_GuardCondition_delete** (p.1376), **DDS_WaitSet_delete** (p.1381), **NDDS_Config_Logger_get_instance** (p.1448), **NDDS_Config_Logger_get_output_device** (p.1456), **NDDS_Config_Logger_set_output_device** (p.1457), **DDS_TypeCodeFactory_finalize_instance** (p.289), **NDDS_Utility_enable_network_capture**, or **NDDS_Utility_disable_network_capture**.

5.212.5.3 NDDS_Config_Logger_get_verbosity()

```
NDDS_Config_LogVerbosity NDDS_Config_Logger_get_verbosity (
    const NDDS_Config_Logger * self )
```

Get the verbosity at which RTI Connex is currently logging diagnostic information.

The default verbosity if **NDDS_Config_Logger_set_verbosity** (p.1450) is never called is **NDDS_CONFIG_LOG_VERBOSITY_ERROR** (p.1443).

If **NDDS_Config_Logger_set_verbosity_by_category** (p.1450) has been used to set different verbosity levels for different categories of messages, this function will return the maximum verbosity of all categories.

5.212.5.4 `NDDS_Config_Logger_get_verbosity_by_category()`

```
NDDS_Config_LogVerbosity NDDS_Config_Logger_get_verbosity_by_category (
    const NDDS_Config_Logger * self,
    NDDS_Config_LogCategory category )
```

Get the verbosity at which RTI Connex is currently logging diagnostic information in the given category.

The default verbosity if `NDDS_Config_Logger_set_verbosity` (p. 1450) and `NDDS_Config_Logger_set_verbosity_by_category` (p. 1450) are never called is `NDDS_CONFIG_LOG_VERBOSITY_ERROR` (p. 1443).

5.212.5.5 `NDDS_Config_Logger_set_verbosity()`

```
void NDDS_Config_Logger_set_verbosity (
    NDDS_Config_Logger * self,
    NDDS_Config_LogVerbosity verbosity )
```

Set the verbosity at which RTI Connex will log diagnostic information.

Note: Logging at high verbosity levels will be detrimental to your application's performance. Your default setting should typically remain at `NDDS_CONFIG_LOG_VERBOSITY_WARNING` (p. 1443) or below. (The default verbosity if you never set it is `NDDS_CONFIG_LOG_VERBOSITY_ERROR` (p. 1443).)

5.212.5.6 `NDDS_Config_Logger_set_verbosity_by_category()`

```
void NDDS_Config_Logger_set_verbosity_by_category (
    NDDS_Config_Logger * self,
    NDDS_Config_LogCategory category,
    NDDS_Config_LogVerbosity verbosity )
```

Set the verbosity at which RTI Connex will log diagnostic information in the given category.

5.212.5.7 `NDDS_Config_Logger_get_output_file()`

```
FILE * NDDS_Config_Logger_get_output_file (
    NDDS_Config_Logger * self )
```

Get the file to which the logged output is redirected.

If no output file has been registered through `NDDS_Config_Logger_set_output_file` (p. 1450), this function will return `NULL`. In this case, logged output will on most platforms go to standard out as if through `printf`.

5.212.5.8 `NDDS_Config_Logger_set_output_file()`

```
DDS_Boolean NDDS_Config_Logger_set_output_file (
    NDDS_Config_Logger * self,
    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 Connexx 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 `NDDS_Config_Logger_set_output_device` (p. 1457) to configure a custom logging device.

5.212.5.9 `NDDS_Config_Logger_set_output_file_name()`

```
DDS_Boolean NDDS_Config_Logger_set_output_file_name (
    NDDS_Config_Logger * self,
    const char * file_name )
```

Set the name of the file to which the logged output is redirected.

The name may be NULL, in which case further logged output will be redirected to the platform-specific default output location (standard out on most platforms).

See `NDDS_Config_Logger_set_output_file` (p. 1450) for the flushing behavior.

5.212.5.10 `NDDS_Config_Logger_set_output_file_set()`

```
DDS_Boolean NDDS_Config_Logger_set_output_file_set (
    NDDS_Config_Logger * self,
    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. 2109), 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 `NDDS_Config_Logger_set_output_file` (p. 1450) for the flushing behavior.

5.212.5.11 `NDDS_Config_Logger_get_print_format()`

```
NDDS_Config_LogPrintFormat NDDS_Config_Logger_get_print_format (
    const NDDS_Config_Logger * self )
```

Get the current message format for the log level **NDDS_CONFIG_LOG_LEVEL_ERROR** (p. 1444).

Use **NDDS_Config_Logger_get_print_format_by_log_level** (p. 1452) to retrieve the format for other log levels.

If **NDDS_Config_Logger_set_print_format** (p. 1452) is never called, the default format is **NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT** (p. 1446).

5.212.5.12 `NDDS_Config_Logger_get_print_format_by_log_level()`

```
NDDS_Config_LogPrintFormat NDDS_Config_Logger_get_print_format_by_log_level (
    const NDDS_Config_Logger * self,
    NDDS_Config_LogLevel log_level )
```

Get the current message format, by log level, that RTI Connexx is using to log diagnostic information.

If **NDDS_Config_Logger_set_print_format** (p. 1452) is never called, the default format is **NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT** (p. 1446).

5.212.5.13 `NDDS_Config_Logger_set_print_format()`

```
DDS_Boolean NDDS_Config_Logger_set_print_format (
    NDDS_Config_Logger * self,
    NDDS_Config_LogPrintFormat print_format )
```

Set the message format that RTI Connexx will use to log diagnostic information for all the log levels, except for **NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR** (p. 1444). When the **Activity Context** (p. 1458) is printed, the user can select the information that will be part of the **Activity Context** (p. 1458) by using the API **NDDS_Config_ActivityContext_set_attribute_mask** (p. 1462).

Set print mask for all the log levels, except for (**RTI_LOG_LEVEL_FATAL_ERROR**).

5.212.5.14 `NDDS_Config_Logger_set_print_format_by_log_level()`

```
DDS_Boolean NDDS_Config_Logger_set_print_format_by_log_level (
    NDDS_Config_Logger * self,
    NDDS_Config_LogPrintFormat print_format,
    NDDS_Config_LogLevel log_level )
```

Set the message format that RTI Connexx will use to log diagnostic information for all the log levels, except for **NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR** (p. 1444). When the **Activity Context** (p. 1458) is printed, the user can select the information that will be part of the **Activity Context** (p. 1458) by using the API **NDDS_Config_ActivityContext_set_attribute_mask** (p. 1462).

5.212.5.15 NDDS_Config_Logger_emergency()

```
void NDDS_Config_Logger_emergency (
    NDDS_Config_Logger * self,
    const char * msg )
```

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_EMERGENCY** (p.1444) and **NDDS_CONFIG_LOG_CATEGORY_USER** (p.1446). **Note:** **DDS_DomainParticipantFactory_finalize_instance** (p.38) needs to be called to finalize the Logger singleton. .

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this function while another thread may be simultaneously calling **DDS_DomainParticipantFactory_get_instance** (p.38), **DDS_DomainParticipantFactory_finalize_instance** (p.38), **DDS_TypeCodeFactory_get_instance** (p.289), **DDS_GuardCondition_new** (p.1376), **DDS_WaitSet_new** (p.1380), **DDS_WaitSet_new_ex** (p.1380), **DDS_GuardCondition_delete** (p.1376), **DDS_WaitSet_delete** (p.1381), **NDDS_Config_Logger_get_instance** (p.1448), **NDDS_Config_Logger_get_output_device** (p.1456), **NDDS_Config_Logger_set_output_device** (p.1457), **DDS_TypeCodeFactory_finalize_instance** (p.289), **NDDS_Utility_enable_network_capture**, or **NDDS_Utility_disable_network_capture**.

5.212.5.16 NDDS_Config_Logger_alert()

```
void NDDS_Config_Logger_alert (
    NDDS_Config_Logger * self,
    const char * msg )
```

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_ALERT** (p.1444) and **NDDS_CONFIG_LOG_CATEGORY_USER** (p.1446). **Note:** **DDS_DomainParticipantFactory_finalize_instance** (p.38) needs to be called to finalize the Logger singleton. .

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this function while another thread may be simultaneously calling **DDS_DomainParticipantFactory_get_instance** (p.38), **DDS_DomainParticipantFactory_finalize_instance** (p.38), **DDS_TypeCodeFactory_get_instance** (p.289), **DDS_GuardCondition_new** (p.1376), **DDS_WaitSet_new** (p.1380), **DDS_WaitSet_new_ex** (p.1380), **DDS_GuardCondition_delete** (p.1376), **DDS_WaitSet_delete** (p.1381), **NDDS_Config_Logger_get_instance** (p.1448), **NDDS_Config_Logger_get_output_device** (p.1456), **NDDS_Config_Logger_set_output_device** (p.1457), **DDS_TypeCodeFactory_finalize_instance** (p.289), **NDDS_Utility_enable_network_capture**, or **NDDS_Utility_disable_network_capture**.

5.212.5.17 NDDS_Config_Logger_critical()

```
void NDDS_Config_Logger_critical (
    NDDS_Config_Logger * self,
    const char * msg )
```

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_CRITICAL** (p.1445) and **NDDS_CONFIG_LOG_CATEGORY_USER** (p.1446). **Note:** **DDS_DomainParticipantFactory_finalize_instance** (p.38) needs to be called to finalize the Logger singleton. .

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this function while another thread may be simultaneously calling **DDS_DomainParticipantFactory_get_instance** (p.38), **DDS_DomainParticipantFactory_finalize_instance** (p.38), **DDS_TypeCodeFactory_get_instance** (p.289), **DDS_GuardCondition_new** (p.1376), **DDS_WaitSet_new** (p.1380), **DDS_WaitSet_new_ex** (p.1380), **DDS_GuardCondition_delete** (p.1376), **DDS_WaitSet_delete** (p.1381), **NDDS_Config_Logger_get_instance** (p.1448), **NDDS_Config_Logger_get_output_device** (p.1456), **NDDS_Config_Logger_set_output_device** (p.1457), **DDS_TypeCodeFactory_finalize_instance** (p.289), **NDDS_Utility_enable_network_capture**, or **NDDS_Utility_disable_network_capture**.

5.212.5.18 NDDS_Config_Logger_error()

```
void NDDS_Config_Logger_error (
    NDDS_Config_Logger * self,
    const char * msg )
```

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_ERROR** (p.1445) and **NDDS_CONFIG_LOG_CATEGORY_USER** (p.1446). **Note:** **DDS_DomainParticipantFactory_finalize_instance** (p.38) needs to be called to finalize the Logger singleton. .

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this function while another thread may be simultaneously calling **DDS_DomainParticipantFactory_get_instance** (p.38), **DDS_DomainParticipantFactory_finalize_instance** (p.38), **DDS_TypeCodeFactory_get_instance** (p.289), **DDS_GuardCondition_new** (p.1376), **DDS_WaitSet_new** (p.1380), **DDS_WaitSet_new_ex** (p.1380), **DDS_GuardCondition_delete** (p.1376), **DDS_WaitSet_delete** (p.1381), **NDDS_Config_Logger_get_instance** (p.1448), **NDDS_Config_Logger_get_output_device** (p.1456), **NDDS_Config_Logger_set_output_device** (p.1457), **DDS_TypeCodeFactory_finalize_instance** (p.289), **NDDS_Utility_enable_network_capture**, or **NDDS_Utility_disable_network_capture**.

5.212.5.19 NDDS_Config_Logger_warning()

```
void NDDS_Config_Logger_warning (
    NDDS_Config_Logger * self,
    const char * msg )
```

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_WARNING** (p.1445) and **NDDS_CONFIG_LOG_CATEGORY_USER** (p.1446). **Note:** **DDS_DomainParticipantFactory_finalize_instance** (p.38) needs to be called to finalize the Logger singleton. .

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this function while another thread may be simultaneously calling **DDS_DomainParticipantFactory_get_instance** (p.38), **DDS_DomainParticipantFactory_finalize_instance** (p.38), **DDS_TypeCodeFactory_get_instance** (p.289), **DDS_GuardCondition_new** (p.1376), **DDS_WaitSet_new** (p.1380), **DDS_WaitSet_new_ex** (p.1380), **DDS_GuardCondition_delete** (p.1376), **DDS_WaitSet_delete** (p.1381), **NDDS_Config_Logger_get_instance** (p.1448), **NDDS_Config_Logger_get_output_device** (p.1456), **NDDS_Config_Logger_set_output_device** (p.1457), **DDS_TypeCodeFactory_finalize_instance** (p.289), **NDDS_Utility_enable_network_capture**, or **NDDS_Utility_disable_network_capture**.

5.212.5.20 NDDS_Config_Logger_notice()

```
void NDDS_Config_Logger_notice (
    NDDS_Config_Logger * self,
    const char * msg )
```

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_NOTICE** (p.1445) and **NDDS_CONFIG_LOG_CATEGORY_USER** (p.1446). **Note:** **DDS_DomainParticipantFactory_finalize_instance** (p.38) needs to be called to finalize the Logger singleton. .

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this function while another thread may be simultaneously calling **DDS_DomainParticipantFactory_get_instance** (p.38), **DDS_DomainParticipantFactory_finalize_instance** (p.38), **DDS_TypeCodeFactory_get_instance** (p.289), **DDS_GuardCondition_new** (p.1376), **DDS_WaitSet_new** (p.1380), **DDS_WaitSet_new_ex** (p.1380), **DDS_GuardCondition_delete** (p.1376), **DDS_WaitSet_delete** (p.1381), **NDDS_Config_Logger_get_instance** (p.1448), **NDDS_Config_Logger_get_output_device** (p.1456), **NDDS_Config_Logger_set_output_device** (p.1457), **DDS_TypeCodeFactory_finalize_instance** (p.289), **NDDS_Utility_enable_network_capture**, or **NDDS_Utility_disable_network_capture**.

5.212.5.21 NDDS_Config_Logger_informational()

```
void NDDS_Config_Logger_informational (
    NDDS_Config_Logger * self,
    const char * msg )
```

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_INFORMATIONAL** (p.1445) and **NDDS_CONFIG_LOG_CATEGORY_USER** (p.1446). **Note:** **DDS_DomainParticipantFactory_finalize_instance** (p.38) needs to be called to finalize the Logger singleton. .

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this function while another thread may be simultaneously calling **DDS_DomainParticipantFactory_get_instance** (p.38), **DDS_DomainParticipantFactory_finalize_instance** (p.38), **DDS_TypeCodeFactory_get_instance** (p.289), **DDS_GuardCondition_new** (p.1376), **DDS_WaitSet_new** (p.1380), **DDS_WaitSet_new_ex** (p.1380), **DDS_GuardCondition_delete** (p.1376), **DDS_WaitSet_delete** (p.1381), **NDDS_Config_Logger_get_instance** (p.1448), **NDDS_Config_Logger_get_output_device** (p.1456), **NDDS_Config_Logger_set_output_device** (p.1457), **DDS_TypeCodeFactory_finalize_instance** (p.289), **NDDS_Utility_enable_network_capture**, or **NDDS_Utility_disable_network_capture**.

5.212.5.22 NDDS_Config_Logger_debug()

```
void NDDS_Config_Logger_debug (
    NDDS_Config_Logger * self,
    const char * msg )
```

Logs message with **NDDS_CONFIG_SYSLOG_LEVEL_DEBUG** (p.1445) and **NDDS_CONFIG_LOG_CATEGORY_USER** (p.1446). **Note:** **DDS_DomainParticipantFactory_finalize_instance** (p.38) needs to be called to finalize the Logger singleton. .

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this function while another thread may be simultaneously calling **DDS_DomainParticipantFactory_get_instance** (p.38), **DDS_DomainParticipantFactory_finalize_instance** (p.38), **DDS_TypeCodeFactory_get_instance** (p.289), **DDS_GuardCondition_new** (p.1376), **DDS_WaitSet_new** (p.1380), **DDS_WaitSet_new_ex** (p.1380), **DDS_GuardCondition_delete** (p.1376), **DDS_WaitSet_delete** (p.1381), **NDDS_Config_Logger_get_instance** (p.1448), **NDDS_Config_Logger_get_output_device** (p.1456), **NDDS_Config_Logger_set_output_device** (p.1457), **DDS_TypeCodeFactory_finalize_instance** (p.289), **NDDS_Utility_enable_network_capture**, or **NDDS_Utility_disable_network_capture**.

5.212.5.23 NDDS_Config_Logger_get_output_device()

```
struct NDDS_Config_LoggerDevice * NDDS_Config_Logger_get_output_device (
    NDDS_Config_Logger * self )
```

Return the user device registered with the logger.

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this function while another thread may be simultaneously calling **DDS_DomainParticipantFactory_get_instance** (p.38), **DDS_DomainParticipantFactory_finalize_instance** (p.38), **DDS_TypeCodeFactory_get_instance** (p.289), **DDS_GuardCondition_new** (p.1376), **DDS_WaitSet_new** (p.1380), **DDS_WaitSet_new_ex** (p.1380), **DDS_GuardCondition_delete** (p.1376), **DDS_WaitSet_delete** (p.1381), **NDDS_Config_Logger_get_instance** (p.1448), **NDDS_Config_Logger_get_output_device** (p.1456), **NDDS_Config_Logger_set_output_device** (p.1457), **DDS_TypeCodeFactory_finalize_instance** (p.289), **NDDS_Utility_enable_network_capture**, or **NDDS_Utility_disable_network_capture**.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

Registered user device or NULL if no user device is registered.

5.212.5.24 NDDS_Config_Logger_set_output_device()

```
DDS_Boolean NDDS_Config_Logger_set_output_device (
    NDDS_Config_Logger * self,
    struct NDDS_Config_LoggerDevice * device )
```

Register a **NDDS_Config_LoggerDevice** (p.2116).

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 **NDDS_Config_Logger_set_output_file** (p.1450).

To remove an existing device, use this function 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 function providing the new device as the device parameter.

When a device is unregistered (by setting it to NULL), **NDDS_Config_LoggerDevice** (p.2116) calls the function **NDDS_Config_LoggerDevice::close** (p.2117).

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this function while another thread may be simultaneously calling **DDS_DomainParticipantFactory_get_instance** (p. 38), **DDS_DomainParticipantFactory_finalize_instance** (p. 38), **DDS_TypeCodeFactory_get_instance** (p. 289), **DDS_GuardCondition_new** (p. 1376), **DDS_WaitSet_new** (p. 1380), **DDS_WaitSet_new_ex** (p. 1380), **DDS_GuardCondition_delete** (p. 1376), **DDS_WaitSet_delete** (p. 1381), **NDDS_Config_Logger_get_instance** (p. 1448), **NDDS_Config_Logger_get_output_device** (p. 1456), **NDDS_Config_Logger_set_output_device** (p. 1457), **DDS_TypeCodeFactory_finalize_instance** (p. 289), **NDDS_Utility_enable_network_capture**, or **NDDS_Utility_disable_network_capture**.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>device</i>	<< <i>in</i> >> (p. 1013) Logging device.

5.213 Activity Context

Add contextual information to log messages.

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. 1460) 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. 1458) 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

- void **NDDS_Config_ActivityContext_set_attribute_mask** (**NDDS_Config_ActivityContextAttributeKind**↔ **Mask** attribute_mask)

*Set the **NDDS_Config_ActivityContextAttributeKindMask** (p. 1460) of the Activity Context.*

5.213.1 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 Connex logging infrastructure supports. It is used by default in **NDDS**↔ **_CONFIG_LOG_PRINT_FORMAT_DEFAULT** (p. 1446). 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. 1446) DDS logging infrastructure supports. If that format is set every time RTI Connex 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,0X79E5D71,0X50EE914:0X1C1:0X800000088{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,0X79E5D71,0X50EE914:0X1C1:0X800000003{Name=testDataWriterName,Entity=DW,Topic=test,Type=Foo,Domain=1}|Write]
```

5.213.2 Macro Definition Documentation

5.213.2.1 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.

5.213.2.2 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.

5.213.2.3 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.

5.213.3 Typedef Documentation

5.213.3.1 NDDS_Config_ActivityContextAttributeKindMask

```
typedef DDS_Long NDDS_Config_ActivityContextAttributeKindMask
```

The attributes **NDDS_Config_ActivityContextAttributeKind** (p. 1460) 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 Connex logs a message or when the Heap Monitoring utility saves statistics for a memory allocation.

5.213.4 Enumeration Type Documentation

5.213.4.1 NDDS_Config_ActivityContextAttributeKind

```
enum NDDS_Config_ActivityContextAttributeKind
```

The resources of the **Activity Context** (p. 1458) 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=testDataWriterName,Entity=DW,Topic=test,Type=Foo}] - The GUID prefix is 0X101A76B,0X79E5D71,0X50EE914. If the bit NDDS_CONFIG_ACTIVITY_CONTEXT_↔ ATTRIBUTE_GUID_PREFIX (p. 1461) 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=testDataWriterName,Entity=DW,Topic=test,Type=Foo}] - The Topic is "test." If the bit NDDS_CONFIG_ACTIVITY_CONTEXT_↔ ATTRIBUTE_TOPIC (p. 1461) is not set, the string representation will not show the Topic: [0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Type=Foo}]
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=testDataWriterName,Entity=DW,Topic=test,Type=Foo}] - The data type is "Foo." If the bit NDDS_CONFIG_ACTIVITY_CONTEXT_↔ ATTRIBUTE_TYPE (p. 1461) is not set, the string representation will not show the data type: [0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName,Entity=DW,Topic=test}]

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=testDataWriterName}] - The entity kind is "Writer." If the bit NDDS_CONFIG_ACTIVITY_CONTEXT_↔ ATTRIBUTE_ENTITY_KIND (p. 1462) is not set, the string representation will not show the entity kind: [0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName}]
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=testDataWriterName}] - The domain ID is "1." If the bit NDDS_CONFIG_ACTIVITY_CONTEXT_↔ ATTRIBUTE_DOMAIN_ID (p. 1462) is not set, the string representation will not show the domain ID: [0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Name=testDataWriterName}]
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=testDataWriterName}] - The entity name is "testDataWriterName." If the bit NDDS_CONFIG_ACTIVITY_CONTEXT_↔ ATTRIBUTE_ENTITY_NAME (p. 1462) is not set, the string representation will not show the entity name: [0X101A76B,0X79E5D71,0X50EE914:0X1C1:0X80000003{Entity=DW,Topic=testDataWriterName}]

5.213.5 Function Documentation

5.213.5.1 `NDDS_Config_ActivityContext_set_attribute_mask()`

```
void NDDS_Config_ActivityContext_set_attribute_mask (
    NDDS_Config_ActivityContextAttributeKindMask attribute_mask )
```

Set the `NDDS_Config_ActivityContextAttributeKindMask` (p. 1460) of the Activity Context.

5.214 Compliance Configuration

APIs to configure compliance with certain standard specifications.

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 }`
The bits for the `NDDS_Config_XTypesComplianceMask` (p. 1465) 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. 1466) with different aspects of RPC and Request Reply.

Functions

- **NDDS_Config_XTypesComplianceMask NDDSD_Config_get_xtypes_compliance_mask** (void)
Gets the XTypes compliance mask value for the application.
- void **NDDS_Config_set_xtypes_compliance_mask** (NDDSD_Config_XTypesComplianceMask mask)
Sets the level of compliance of the application with the XTypes specification.
- **DDS_Boolean NDDSD_Config_load_compliance_masks** (void)
Load the compliance masks from the environment variables.
- **NDDS_Config_RpcComplianceMask NDDSD_Config_get_rpc_compliance_mask** (void)
Gets the RPC compliance mask value for the application.
- void **NDDS_Config_set_rpc_compliance_mask** (NDDSD_Config_RpcComplianceMask mask)
Sets the level of compliance of the application with the DDS RPC specification.

5.214.1 Detailed Description

APIs to configure compliance with certain standard specifications.

5.214.2 Macro Definition Documentation

5.214.2.1 NDDSD_CONFIG_XTYPES_COMPLIANCE_MASK_VENDOR

```
#define NDDSD_CONFIG_XTYPES_COMPLIANCE_MASK_VENDOR
```

XTypes vendor compliance mask. This value is fully aligned with the XTypes specification.

The mask sets the following bits:

- **NDDSD_CONFIG_XTYPES_DHEADER_IN_NON_PRIMITIVE_COLLECTIONS_BIT** (p. 1468)
- **NDDSD_CONFIG_XTYPES_ENCAPSULATION_OPTIONS_WITH_PADDING_BIT** (p. 1469)
- **NDDSD_CONFIG_XTYPES_ACCEPT_UNKNOWN_DISCRIMINATOR_BIT** (p. 1470)
- **NDDSD_CONFIG_XTYPES_SENTINEL_IN_EMPTY_UNION_BIT** (p. 1471)
- **NDDSD_CONFIG_XTYPES_INITIALIZE_DISCRIMINATOR_TO_DEFAULT_BIT** (p. 1472)

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.

5.214.2.2 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. 1468)
- **NDDS_CONFIG_XTYPES_ENCAPSULATION_OPTIONS_WITH_PADDING_BIT** (p. 1469)
- **NDDS_CONFIG_XTYPES_SENTINEL_IN_EMPTY_UNION_BIT** (p. 1471)
- **NDDS_CONFIG_XTYPES_INITIALIZE_DISCRIMINATOR_TO_DEFAULT_BIT** (p. 1472)

The HEX value for the mask is: 0x0000018C.

5.214.2.3 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. 1474)

The HEX value for the mask is: 0x00000001.

5.214.2.4 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.

5.214.3 Typedef Documentation

5.214.3.1 **NDDS_Config_XTypesComplianceMask**

```
typedef DDS_UnsignedLong NDDS_Config_XTypesComplianceMask
```

XTypes compliance mask.

The **NDDS_Config_XTypesComplianceMask** (p. 1465) 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 **NDDS_Config_set_xtypes_compliance_mask** (p. 1474)

The environment variable will be automatically loaded when a **DDS_DomainParticipantFactory** (p. 32) is created. If you need to load it before that, use the function **NDDS_Config_load_compliance_masks** (p. 1475).

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. 1464).

To be fully compliant with the XTypes specification, set the mask to **NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_VENDOR** (p. 1464).

[default] NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_DEFAULT (p. 1464)

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.

5.214.3.2 **NDDS_Config_RpcComplianceMask**

```
typedef DDS_UnsignedLong NDDS_Config_RpcComplianceMask
```

Mask to configure compliance with the OMG DDS-RPC specification.

The **NDDS_Config_RpcComplianceMask** (p. 1466) 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 **NDDS_Config_set_rpc_compliance_mask** (p. 1476)

The environment variable will be automatically loaded when a **DDS_DomainParticipantFactory** (p. 32) is created. If you need to load it before that, use the function **NDDS_Config_load_compliance_masks** (p. 1475).

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. 1465).

To be fully compliant with the DDS RPC specification, set the mask to **NDDS_CONFIG_RPC_COMPLIANCE_MASK_↔_VENDOR** (p. 1465).

[default] NDDS_CONFIG_RPC_COMPLIANCE_MASK_DEFAULT (p. 1465)

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 Connex Core Libraries Users Manual.

5.214.4 Enumeration Type Documentation

5.214.4.1 NDDS_Config_XTypesComplianceMaskBits

```
enum NDDS_Config_XTypesComplianceMaskBits
```

The bits for the **NDDS_Config_XTypesComplianceMask** (p. 1465) with different aspects of the Extended CDR encoding (XCDR and XCDR2).

See also

NDDS_Config_XTypesComplianceMask (p. 1465)

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. 1464)
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. 1468) 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. 1464)
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. 1464)

Enumerator

NDDS_CONFIG_XTYPES_ENCAPSULATION_↔ OPTIONS_WITH_PADDING_BIT	When this bit is set, RTI Connex Micro 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 Connex Micro 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 Connex Micro 3.x.y.z • RTI Connex Micro 2.4.15.1 • RTI Connex Micro 2.4.14.0 and 2.4.14.1 • RTI Connex Micro 2.4.13.1 through 2.4.13.5 • RTI Connex 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. 1464)
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. 1464)

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. 1470) 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. 1464)
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. 1472) determines what the default discriminator value is. This bit is only relevant when the bit NDDS_CONFIG_XTYPES_ACCEPT_UNKNOWN_↔ DISCRIMINATOR_BIT (p. 1470) 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. 1464)

Enumerator

<code>NDDS_CONFIG_XTYPES_SENTINEL_IN_EMPTY_UNION_BIT</code>	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 Connex version 5.3.x, this bit must be set. This bit is set by default. See <code>NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_DEFAULT</code> (p. 1464)

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 <i>Command-Line Arguments for rtiddsgen</i>, in the RTI Code Generator User's Manual. This bit affects NDDS_CONFIG_XTYPES_SELECT_↔ DEFAULT_DISCRIMINATOR_BIT (p. 1470). (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. 1464)

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 Connex 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 Connex 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. 1464)
NDDS_CONFIG_XTYPES_PYTHON_INT8_730_BIT	Set this bit to interoperate with Connex 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 Connex 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 Connex 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. 1464)

5.214.4.2 NDDS_Config_RpcComplianceMaskBits

```
enum NDDS_Config_RpcComplianceMaskBits
```

The bits for the **NDDS_Config_RpcComplianceMask** (p. 1466) 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. 1474) will affect both RPC and Request-Reply communications. 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 Connex. This bit is unset by default. See NDDS_CONFIG_↔ RPC_COMPLIANCE_MASK_DEFAULT (p. 1465)

5.214.5 Function Documentation

5.214.5.1 NDDS_Config_get_xtypes_compliance_mask()

```
NDDS_Config_XTypesComplianceMask NDDS_Config_get_xtypes_compliance_mask (
    void )
```

Gets the XTypes compliance mask value for the application.

The **NDDS_Config_XTypesComplianceMask** (p. 1465) configures different aspects of the Extended CDR encoding (XCDR and XCDR2) for the application.

Returns

The **NDDS_Config_XTypesComplianceMask** (p. 1465) value.

See also

NDDS_Config_XTypesComplianceMask (p. 1465)
NDDS_Config_get_xtypes_compliance_mask (p. 1474)
NDDS_Config_load_compliance_masks (p. 1475)

5.214.5.2 NDDS_Config_set_xtypes_compliance_mask()

```
void NDDS_Config_set_xtypes_compliance_mask (
    NDDS_Config_XTypesComplianceMask mask )
```

Sets the level of compliance of the application with the XTypes specification.

The **NDDS_Config_XTypesComplianceMask** (p. 1465) configures different aspects of the Extended CDR encoding (XCDR and XCDDR2) 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

<i>mask</i>	<< <i>in</i> >> (p. 1013) Compliance mask.

See also

- NDDS_Config_XTypesComplianceMask** (p. 1465)
- NDDS_Config_get_xtypes_compliance_mask** (p. 1474)
- NDDS_Config_load_compliance_masks** (p. 1475)

5.214.5.3 NDDS_Config_load_compliance_masks()

```
DDS_Boolean NDDS_Config_load_compliance_masks (
    void )
```

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. 1465)
- NDDS_Config_set_xtypes_compliance_mask** (p. 1474)
- NDDS_Config_get_xtypes_compliance_mask** (p. 1474)
- NDDS_Config_RpcComplianceMask** (p. 1466)
- NDDS_Config_set_rpc_compliance_mask** (p. 1476)
- NDDS_Config_get_rpc_compliance_mask** (p. 1475)

5.214.5.4 `NDDS_Config_get_rpc_compliance_mask()`

```
NDDS_Config_RpcComplianceMask NDDS_Config_get_rpc_compliance_mask (
    void )
```

Gets the RPC compliance mask value for the application.

The **NDDS_Config_RpcComplianceMask** (p. 1466) configures different aspects of RPC communication.

Returns

The **NDDS_Config_RpcComplianceMask** (p. 1466) value.

See also

NDDS_Config_RpcComplianceMask (p. 1466)

NDDS_Config_get_rpc_compliance_mask (p. 1475)

NDDS_Config_load_compliance_masks (p. 1475)

5.214.5.5 `NDDS_Config_set_rpc_compliance_mask()`

```
void NDDS_Config_set_rpc_compliance_mask (
    NDDS_Config_RpcComplianceMask mask )
```

Sets the level of compliance of the application with the DDS RPC specification.

The **NDDS_Config_RpcComplianceMask** (p. 1466) 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 Connex API. The mask can also be set using the environment variable **NDDS_RPC_COMPLIANCE_MASK**.

Parameters

<i>mask</i>	<< <i>in</i> >> (p. 1013) Compliance mask.

See also

NDDS_Config_RpcComplianceMask (p. 1466)

NDDS_Config_get_rpc_compliance_mask (p. 1475)

NDDS_Config_load_compliance_masks (p. 1475)

5.215 Heap Monitoring

Monitor memory allocations done by the middleware on the native heap.

Data Structures

- struct **NDDS_Utility_HeapMonitoringParams_t**
Input parameters for enabling heap monitoring. They will be used for configuring the format of the snapshot.

Macros

- #define **NDDS_Utility_HeapMonitoringParams_INITIALIZER**
*Static initializer for **NDDS_Utility_HeapMonitoringParams_t** (p. 2164).*

Typedefs

- typedef struct **NDDS_Utility_HeapMonitoringParams_t** **NDDS_Utility_HeapMonitoringParams_t**
Input parameters for enabling heap monitoring. They will be used for configuring the format of the snapshot.

Enumerations

- enum **NDDS_Utility_HeapMonitoringSnapshotOutputFormat** {
 NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_OUTPUT_FORMAT_STANDARD ,
 NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_OUTPUT_FORMAT_COMPRESSED }
Specify the format of the output of the snapshot. RTI Connex.
- 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.

Functions

- **DDS_Boolean NDDUtility_enable_heap_monitoring** (void)
Starts monitoring the heap memory used by RTI Connex.
- **DDS_Boolean NDDUtility_enable_heap_monitoring_w_params** (const **NDDS_Utility_HeapMonitoringParams_t** *params)
Starts monitoring the heap memory used by RTI Connex.
- **void NDDUtility_disable_heap_monitoring** (void)
Stops monitoring the heap memory used by RTI Connex.
- **DDS_Boolean NDDUtility_pause_heap_monitoring** (void)
Pauses heap monitoring.
- **DDS_Boolean NDDUtility_resume_heap_monitoring** (void)
Resumes heap monitoring.
- **DDS_Boolean NDDUtility_take_heap_snapshot** (const char *filename, **DDS_Boolean** print_details)
Saves the current heap memory usage in a file.

5.215.1 Detailed Description

Monitor memory allocations done by the middleware on the native heap.

RTI Connexx 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 `NDDS_Utility_enable_heap_monitoring` is called, you may invoke **`NDDS_Utility_take_heap_snapshot`** (p. 1481) 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.

5.215.2 Macro Definition Documentation

5.215.2.1 `NDDS_Utility_HeapMonitoringParams_INITIALIZER`

```
#define NDDS_Utility_HeapMonitoringParams_INITIALIZER
```

Static initializer for **`NDDS_Utility_HeapMonitoringParams_t`** (p. 2164).

Use this initializer to ensure that new objects do not have uninitialized contents.

```
struct NDDS_Utility_HeapMonitoringParams_t param = NDDS_Utility_HeapMonitoringParams_INITIALIZER;
```

See also

`NDDS_Utility_HeapMonitoringParams_t` (p. 2164)

5.215.3 Typedef Documentation

5.215.3.1 `NDDS_Utility_HeapMonitoringParams_t`

```
typedef struct NDDS_Utility_HeapMonitoringParams_t NDDS_Utility_HeapMonitoringParams_t
```

Input parameters for enabling heap monitoring. They will be used for configuring the format of the snapshot.

5.215.4 Enumeration Type Documentation

5.215.4.1 `NDDS_Utility_HeapMonitoringSnapshotOutputFormat`

```
enum NDDS_Utility_HeapMonitoringSnapshotOutputFormat
```

Specify the format of the output of the snapshot. RTI Connexx.

Enumerator

NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT↔ _OUTPUT_FORMAT_STANDARD	The output of the snapshot will be in plain text.
NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT↔ _OUTPUT_FORMAT_COMPRESSED	The output of the snapshot will be compressed using Zlib technology. The file can be uncompressed using zlib-flate.

5.215.4.2 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 NDDS_Config_ActivityContext_set_attribute_mask (p. 1462).
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.

5.215.5 Function Documentation

5.215.5.1 NDDS_Utility_enable_heap_monitoring()

```
DDS_Boolean NDDS_Utility_enable_heap_monitoring (
    void )
```

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 **NDDS_Utility_take_heap_snapshot** (p. 1481).

Use this function only for debugging purposes, since it may introduce a significant performance impact.

MT Safety:

UNSAFE. It is not safe to call this function while another thread may be simultaneously calling another heap-related function, including this one.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201)

See also

NDDS_Utility_disable_heap_monitoring (p. 1480)

5.215.5.2 NDDS_Utility_enable_heap_monitoring_w_params()

```
DDS_Boolean NDDS_Utility_enable_heap_monitoring_w_params (
    const NDDS_Utility_HeapMonitoringParams_t * params )
```

Starts monitoring the heap memory used by RTI Connex.

Performs the same function as **NDDS_Utility_enable_heap_monitoring** except that it also provides the values in *params*. Those values will set the format used in the snapshot **NDDS_Utility_take_heap_snapshot** (p. 1481).

Returns

DDS_BOOLEAN_TRUE (p. 1201) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201)

See also

NDDS_Utility_disable_heap_monitoring (p. 1480)

5.215.5.3 NDDS_Utility_disable_heap_monitoring()

```
void NDDS_Utility_disable_heap_monitoring (
    void )
```

Stops monitoring the heap memory used by RTI Connex.

This function must be the last function called from RTI Connex.

See also

NDDS_Utility_enable_heap_monitoring (p. 1479)

5.215.5.4 NDDS_Utility_pause_heap_monitoring()

```
DDS_Boolean NDDS_Utility_pause_heap_monitoring (
    void )
```

Pauses heap monitoring.

New memory allocations will not be monitored and they will not appear in the snapshot generated by **NDDS_Utility_take_heap_snapshot** (p. 1481).

Returns

DDS_BOOLEAN_TRUE (p. 1201) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201)

See also

NDDS_Utility_resume_heap_monitoring (p. 1481)

5.215.5.5 NDDS_Utility_resume_heap_monitoring()

```
DDS_Boolean NDDS_Utility_resume_heap_monitoring (
    void )
```

Resumes heap monitoring.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201)

See also

NDDS_Utility_pause_heap_monitoring (p. 1480)

5.215.5.6 NDDS_Utility_take_heap_snapshot()

```
DDS_Boolean NDDS_Utility_take_heap_snapshot (
    const char * filename,
    DDS_Boolean print_details )
```

Saves the current heap memory usage in a file.

After `NDDS_Utility_enable_heap_monitoring` is called, you may invoke this function 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 Connex and non-RTI Connex memory.
- Process physical 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 Connex 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 Connex memory, not memory living in the managed heap.
- High watermark: The maximum amount of heap usage by RTI Connex since `NDDS_Utility_enable_heap_monitoring` was invoked.
- Alloc count: The number of invocations to `malloc`, `realloc`, or `calloc` operations done by RTI Connex.
- Free count: The number of invocations to the `free` operation done by RTI Connex.

After the previous summary, and only if you set the parameter `print_details` to **DDS_BOOLEAN_TRUE** (p. 1201), the function will print the details of every single outstanding heap allocation done by RTI Connex. 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 Connex method name.
- `type_name`: The allocation typename.
- `pool_alloc`: Indicates if the heap allocation is a RTI Connex 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. 1458)

Parameters

<code>filename</code>	<< <i>in</i> >> (p. 1013). Name of file in which to store the snapshot.
<code>print_details</code>	<< <i>in</i> >> (p. 1013). Indicates if the snapshot will contain only the memory usage summary or the details of the individual allocations.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201)

5.216 Network Capture

Save network traffic into a capture file for further analysis.

Data Structures

- struct **NDDS_Utility_NetworkCaptureParams_t**
Input parameters for starting network capture.

Macros

- **#define `NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_DEFAULT`**
Default mask for `NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_KIND` (p. 1488): 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_NETWORK_CAPTURE_TRAFFIC_KIND_MASK` (p. 1488).
- **#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_NETWORK_CAPTURE_CONTENT_KIND_MASK`**
Mask that indicates a combination of content types.
- **typedef `DDS_Long NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_KIND_MASK`**
Mask that indicates the traffic direction to capture.
- **typedef struct `NDDS_UTILITY_NETWORK_CAPTURE_PARAMS_t` `NDDS_UTILITY_NETWORK_CAPTURE_PARAMS_t`**
Input parameters for starting network capture.

Enumerations

- **enum `NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_KIND` {**
`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_NETWORK_CAPTURE_TRAFFIC_KIND` {**
`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.

Functions

- **`DDS_Boolean NDDS_UTILITY_enable_network_capture` (void)**
Enable Network Capture.
- **`DDS_Boolean NDDS_UTILITY_disable_network_capture` (void)**
Disable Network Capture.
- **`DDS_Boolean NDDS_UTILITY_set_default_network_capture_params` (const `NDDS_UTILITY_NETWORK_CAPTURE_PARAMS_t` *params)**
Set the default Network Capture parameters.
- **`DDS_Boolean NDDS_UTILITY_start_network_capture` (const char *filename)**
Start capturing traffic for all DomainParticipants, with the default parameters.

- **DDS_Boolean NDDS_Utility_start_network_capture_for_participant** (DDS_DomainParticipant *participant, const char *filename)
Start capturing traffic for a DomainParticipant, with the default parameters.
- **DDS_Boolean NDDS_Utility_start_network_capture_w_params** (const char *filename, const NDDS_Utility_NetworkCaptureParams_t *params)
Start capturing traffic for all DomainParticipants, with the provided parameters.
- **DDS_Boolean NDDS_Utility_start_network_capture_w_params_for_participant** (DDS_DomainParticipant *participant, const char *filename, const NDDS_Utility_NetworkCaptureParams_t *params)
Start capturing traffic for a DomainParticipant, with the provided parameters.
- **DDS_Boolean NDDS_Utility_stop_network_capture** (void)
Stop capturing traffic for all participants.
- **DDS_Boolean NDDS_Utility_stop_network_capture_for_participant** (DDS_DomainParticipant *participant)
Stop capturing traffic for a DomainParticipant.
- **DDS_Boolean NDDS_Utility_pause_network_capture** (void)
Pause capturing traffic for all DomainParticipants.
- **DDS_Boolean NDDS_Utility_pause_network_capture_for_participant** (DDS_DomainParticipant *participant)
Pause capturing traffic for a DomainParticipant.
- **DDS_Boolean NDDS_Utility_resume_network_capture** (void)
Resume capturing traffic for all DomainParticipants.
- **DDS_Boolean NDDS_Utility_resume_network_capture_for_participant** (DDS_DomainParticipant *participant)
Resume capturing traffic for a DomainParticipant.

Variables

- **const NDDS_Utility_NetworkCaptureParams_t NDDS_UTILITY_NETWORK_CAPTURE_PARAMETERS_DEFAULT**
Default parameters used in Network Capture.

5.216.1 Detailed Description

Save network traffic into a capture file for further analysis.

RTI Connexx 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 NDDS_Utility_enable_network_capture must be invoked before creating any DomainParticipant. Similarly, NDDS_Utility_disable_network_capture 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.

5.216.2 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. 1901)).

```
<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.

5.216.3 Macro Definition Documentation

5.216.3.1 NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_DEFAULT

```
#define NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_DEFAULT
```

Default mask for **NDDS_Utility_NetworkCaptureContentKind** (p. 1488): do not remove any content.

It is equivalent to **NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_NONE** (p. 1486).

[default] Do not remove any content.

5.216.3.2 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.

5.216.3.3 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. 1489)) | **NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_ENCRYPTED_DATA** (p. 1489))

5.216.3.4 NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_DEFAULT

```
#define NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_DEFAULT
```

Default mask for **NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK** (p. 1488).

It is equivalent to **NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_ALL** (p. 1487).

[default] Capture all traffic: inbound and outbound.

5.216.3.5 NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_NONE

```
#define NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_NONE
```

Do not capture any traffic.

5.216.3.6 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. 1489)) | **NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_IN** (p. 1489)).

5.216.4 Typedef Documentation

5.216.4.1 `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. 1488) bitmaps.

See also

`NDDS_Utility_NetworkCaptureContentKind` (p. 1488)

5.216.4.2 `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. 1489) bitmaps.

See also

`NDDS_Utility_NetworkCaptureTrafficKind` (p. 1489)

5.216.4.3 `NDDS_Utility_NetworkCaptureParams_t`

```
typedef struct NDDS_Utility_NetworkCaptureParams_t NDDS_Utility_NetworkCaptureParams_t
```

Input parameters for starting network capture.

5.216.5 Enumeration Type Documentation

5.216.5.1 `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. 1488) 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.

5.216.5.2 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. 1488) 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.

5.216.6 Function Documentation

5.216.6.1 NDDS_Utility_enable_network_capture()

```
DDS_Boolean NDDS_Utility_enable_network_capture (
    void )
```

Enable Network Capture.

This function must be called before any other Network Capture function. It must also be called before creating the participants for which we want to capture traffic.

Use this function only for debugging purposes, since it may introduce a significant performance impact.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201)

MT Safety:

UNSAFE. It is not safe to call this function while another thread may be simultaneously calling another Network Capture related function, including this one.

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this function while another thread may be simultaneously calling **DDS_↵DomainParticipantFactory_get_instance** (p. 38), **DDS_DomainParticipantFactory_finalize_instance** (p. 38), **DDS_TypeCodeFactory_get_instance** (p. 289), **DDS_TypeCodeFactory_finalize_instance** (p. 289), **NDDS_↵Utility_enable_network_capture**, or **NDDS_Utility_disable_network_capture**.

See also

NDDS_Utility_disable_network_capture (p. 1490)

5.216.6.2 NDDS_Utility_disable_network_capture()

```
DDS_Boolean NDDS_Utility_disable_network_capture (
    void )
```

Disable Network Capture.

This function must be the last Network Capture function 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. 1201) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201)

MT Safety:

UNSAFE. It is not safe to call this function while another thread may be simultaneously calling another Network Capture related function, including this one.

MT Safety:

UNSAFE. In VxWorks, it is unsafe to call this function while another thread may be simultaneously calling **DDS_↵DomainParticipantFactory_get_instance** (p. 38), **DDS_DomainParticipantFactory_finalize_instance** (p. 38), **DDS_TypeCodeFactory_get_instance** (p. 289), **DDS_TypeCodeFactory_finalize_instance** (p. 289), **NDDS_↵Utility_enable_network_capture**, or **NDDS_Utility_disable_network_capture**.

See also

NDDS_Utility_enable_network_capture (p. 1489)

5.216.6.3 NDDS_Utility_set_default_network_capture_params()

```
DDS_Boolean NDDS_Utility_set_default_network_capture_params (
    const NDDS_Utility_NetworkCaptureParams_t * params )
```

Set the default Network Capture parameters.

The default parameters are used when Network Capture is started without parameters, i.e., `NDDS_Utility_start_network_capture`.

Precondition

This function requires first enabling Network Capture. See `NDDS_Utility_enable_network_capture`.

Parameters

<i>params</i>	<< <i>in</i> >> (p. 1013). Configuration parameters that we want to set as defaults.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201)

See also

NDDS_Utility_start_network_capture (p. 1491)

NDDS_Utility_start_network_capture_for_participant (p. 1492)

5.216.6.4 NDDS_Utility_start_network_capture()

```
DDS_Boolean NDDS_Utility_start_network_capture (
    const char * filename )
```

Start capturing traffic for all DomainParticipants, with the default parameters.

Precondition

This function requires first enabling Network Capture. See `NDDS_Utility_enable_network_capture`.

Parameters

<i>filename</i>	<< <i>in</i> >> (p. 1013). 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. 1201) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201)

See also

NDDS_Utility_stop_network_capture (p. 1494)

NDDS_Utility_start_network_capture_for_participant (p. 1492)

5.216.6.5 NDDS_Utility_start_network_capture_for_participant()

```
DDS_Boolean NDDS_Utility_start_network_capture_for_participant (
    DDS_DomainParticipant * participant,
    const char * filename )
```

Start capturing traffic for a DomainParticipant, with the default parameters.

Precondition

This function requires first enabling Network Capture. See **NDDS_Utility_enable_network_capture**.

Parameters

<i>participant</i>	<< <i>in</i> >> (p. 1013). DomainParticipant for which we want to capture traffic.
<i>filename</i>	<< <i>in</i> >> (p. 1013). 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. 1201) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201)

See also

NDDS_Utility_stop_network_capture_for_participant (p. 1494)

NDDS_Utility_start_network_capture_w_params_for_participant (p. 1493)

NDDS_Utility_enable_network_capture (p. 1489)

5.216.6.6 NDDS_Utility_start_network_capture_w_params()

```
DDS_Boolean NDDS_Utility_start_network_capture_w_params (
    const char * filename,
    const NDDS_Utility_NetworkCaptureParams_t * params )
```

Start capturing traffic for all DomainParticipants, with the provided parameters.

Precondition

This function requires first enabling Network Capture. See `NDDS_Utility_enable_network_capture`.

Performs the same function as `NDDS_Utility_start_network_capture` except that it uses the provided parameters, instead of the default ones.

Parameters

<i>filename</i>	<< <i>in</i> >> (p. 1013). 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

<i>params</i>	<< <i>in</i> >> (p. 1013). Configuration parameters for the capture.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201)

See also

NDDS_Utility_stop_network_capture (p. 1494)

NDDS_Utility_start_network_capture_w_params_for_participant (p. 1493)

5.216.6.7 NDDS_Utility_start_network_capture_w_params_for_participant()

```
DDS_Boolean NDDS_Utility_start_network_capture_w_params_for_participant (
    DDS_DomainParticipant * participant,
    const char * filename,
    const NDDS_Utility_NetworkCaptureParams_t * params )
```

Start capturing traffic for a DomainParticipant, with the provided parameters.

Precondition

This function requires enabling first Network Capture. See `NDDS_Utility_enable_network_capture`.

Parameters

<i>participant</i>	<< <i>in</i> >> (p. 1013). DomainParticipant for which we want to capture traffic.
<i>filename</i>	<< <i>in</i> >> (p. 1013). 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

<i>params</i>	<< <i>in</i> >> (p. 1013). Parameters for configuring the capture.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201)

See also

NDDS_Utility_stop_network_capture_for_participant (p. 1494)

NDDS_Utility_start_network_capture_for_participant (p. 1492)

5.216.6.8 NDDS_Utility_stop_network_capture()

```
DDS_Boolean NDDS_Utility_stop_network_capture (
    void )
```

Stop capturing traffic for all participants.

Precondition

This function requires enabling first Network Capture. See `NDDS_Utility_enable_network_capture`.

This function can (and must) be called after `NDDS_Utility_start_network_capture`, not `NDDS_Utility_start_network_capture_for_participant`. 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. 1201) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201)

See also

NDDS_Utility_start_network_capture (p. 1491)

NDDS_Utility_stop_network_capture_for_participant (p. 1494)

5.216.6.9 NDDS_Utility_stop_network_capture_for_participant()

```
DDS_Boolean NDDS_Utility_stop_network_capture_for_participant (
    DDS_DomainParticipant * participant )
```

Stop capturing traffic for a DomainParticipant.

Precondition

This function requires first enabling Network Capture. See NDDS_Utility_enable_network_capture.

Parameters

<i>participant</i>	<< <i>in</i> >> (p. 1013). DomainParticipant for which we want to stop capturing traffic.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201)

See also

NDDS_Utility_start_network_capture_for_participant (p. 1492)

NDDS_Utility_stop_network_capture (p. 1494)

5.216.6.10 NDDS_Utility_pause_network_capture()

```
DDS_Boolean NDDS_Utility_pause_network_capture (
    void )
```

Pause capturing traffic for all DomainParticipants.

Precondition

This function requires first enabling Network Capture. See NDDS_Utility_enable_network_capture.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201)

See also

NDDS_Utility_resume_network_capture (p. 1496)

NDDS_Utility_pause_network_capture_for_participant (p. 1495)

5.216.6.11 NDDS_Utility_pause_network_capture_for_participant()

```
DDS_Boolean NDDS_Utility_pause_network_capture_for_participant (
    DDS_DomainParticipant * participant )
```

Pause capturing traffic for a DomainParticipant.

Precondition

This function requires first enabling Network Capture. See NDDS_Utility_enable_network_capture.

Parameters

<i>participant</i>	<< <i>in</i> >> (p. 1013). DomainParticipant for which we want to pause capturing traffic.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201)

See also

NDDS_Utility_resume_network_capture_for_participant (p. 1496)

NDDS_Utility_pause_network_capture (p. 1495)

5.216.6.12 NDDS_Utility_resume_network_capture()

```
DDS_Boolean NDDS_Utility_resume_network_capture (
    void )
```

Resume capturing traffic for all DomainParticipants.

Precondition

This function requires first enabling Network Capture. See NDDS_Utility_enable_network_capture.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201)

See also

NDDS_Utility_pause_network_capture (p. 1495)

NDDS_Utility_resume_network_capture_for_participant (p. 1496)

5.216.6.13 NDDS_Utility_resume_network_capture_for_participant()

```
DDS_Boolean NDDS_Utility_resume_network_capture_for_participant (
    DDS_DomainParticipant * participant )
```

Resume capturing traffic for a DomainParticipant.

Precondition

This function requires first enabling Network Capture. See `NDDS_Utility_enable_network_capture`.

Parameters

<i>participant</i>	<< <i>in</i> >> (p. 1013). DomainParticipant for which we want to resume capturing traffic.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if success. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201)

See also

`NDDS_Utility_pause_network_capture_for_participant` (p. 1495)

`NDDS_Utility_resume_network_capture` (p. 1496)

5.216.7 Variable Documentation

5.216.7.1 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 initalizers to ensure that new objects do not have uninitialized contents.

Initialization of a **NDDS_Utility_NetworkCaptureParams_t** (p. 2165) 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 function:

```
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. 2165).

5.217 Other Utilities

Other Utilities, such as **NDDS_Utility_spin** (p. 1499).

Functions

- **DDS_Boolean NDDS_LM_unload_activation_library** (void)
Unloads the Activation Library (rti_activation) if it was previously loaded.
- void **NDDS_Config_enable_automatic_tss_cleanup** (**DDS_Boolean** enable)
Enable automatic thread-specific storage (TSS) cleanup on platforms that support it.
- void **NDDS_Utility_sleep** (const struct **DDS_Duration_t** *durationIn)
Block the calling thread for the specified duration.
- void **NDDS_Utility_spin** (**DDS_UnsignedLongLong** spinCount)
Performs the spin operation as many times as indicated.
- **DDS_UnsignedLongLong NDDS_Utility_get_spin_per_microsecond** (void)
Returns the number of spin operations to perform to wait 1 microsecond.

5.217.1 Detailed Description

Other Utilities, such as **NDDS_Utility_spin** (p. 1499).

5.217.2 Function Documentation

5.217.2.1 NDDS_LM_unload_activation_library()

```
DDS_Boolean NDDS_LM_unload_activation_library (
    void )
```

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. 1201) if successful. Otherwise, **DDS_BOOLEAN_FALSE** (p. 1201)

5.217.2.2 NDDS_Config_enable_automatic_tss_cleanup()

```
void NDDS_Config_enable_automatic_tss_cleanup (
    DDS_Boolean enable )
```

Enable automatic thread-specific storage (TSS) cleanup on platforms that support it.

This API must be called before any other RTI Connex API.

Setting it to one value and then setting it to the opposite value after any RTI Connex API has been called may result in undefined behavior.

See also

DDS_DomainParticipantFactory_unregister_thread (p. 58)

5.217.2.3 NDDS_Utility_sleep()

```
void NDDS_Utility_sleep (
    const struct DDS_Duration_t * durationIn )
```

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

<i>durationIn</i>	<< <i>in</i> >> (p. 1013) Sleep duration. Cannot be NULL.

MT Safety:

safe

Examples

HelloWorld_publisher.c, and **HelloWorld_subscriber.c**.

5.217.2.4 NDDS_Utility_spin()

```
void NDDS_Utility_spin (
    DDS_UnsignedLongLong spinCount )
```

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

<i>spinCount</i>	<< <i>in</i> >> (p. 1013) Number of spin operations to perform.

5.217.2.5 NDDS_Utility_get_spin_per_microsecond()

```
DDS_UnsignedLongLong NDDS_Utility_get_spin_per_microsecond (
    void )
```

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()`.

Returns

Number of spin operations to wait 1 microsecond.

See also

NDDS_Utility_spin (p. 1499)

5.218 Octet Buffer Support

<<*extension*>> (p. 1012) 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.

5.218.1 Detailed Description

<<*extension*>> (p. 1012) Octet buffer creation, cloning, and deletion.

The functions in this class ensure consistent cross-platform implementations for OctetBuffer creation (**DDS_↔OctetBuffer_alloc()** (p. 1502)), deletion (**DDS_OctetBuffer_free()** (p. 1503)), and cloning (**DDS_OctetBuffer_dup()** (p. 1502)) that preserve the mutable value type semantics. These are to be viewed as functions that define an Octet↔Buffer class whose data is represented by a 'unsigned char*'.

5.218.2 Conventions

The following conventions govern the memory management of OctetBuffers in RTI Connex.

- 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 functions.
 - 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 functions.

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 Connex must therefore make some assumptions when a user requests that a OctetBuffer be copied into. The following rules apply when RTI Connex is copying into an OctetBuffer.

- If the 'unsigned char*' is NULL, RTI Connex 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. 1501) 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 Connex will assume that you are managing the OctetBuffer's memory yourself and have allocated enough memory to store the OctetBuffer to be copied. *RTI Connex 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 Connex; you are likely to crash.*

5.218.3 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. 1502), with matching occurrences of **DDS_OctetBuffer_↔free()** (p. 1503) in the code.
- Finalize value types containing OctetBuffer. In C++ the destructor accomplishes this automatically. in C, explicit "destructor" functions are provided.

5.218.4 Function Documentation

5.218.4.1 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 function must be deleted using **DDS_OctetBuffer_free()** (p. 1503).

This function will allocate enough memory to hold an OctetBuffer of `size` octets.

Parameters

<i>size</i>	<< <i>in</i> >> (p. 1013) Size of the buffer.

Returns

A newly created non-NULL OctetBuffer upon success or NULL upon failure.

5.218.4.2 DDS_OctetBuffer_dup()

```
unsigned char * DDS_OctetBuffer_dup (
    const unsigned char * buffer,
    unsigned int size )
```

Clone an OctetBuffer.

An OctetBuffer created by this function must be deleted using `DDS_OctetBuffer_free()`

Parameters

<i>buffer</i>	<< <i>in</i> >> (p. 1013) The OctetBuffer to duplicate.
<i>size</i>	<< <i>in</i> >> (p. 1013) Size of the OctetBuffer to duplicate.

Returns

If `src == NULL` or `size < 0`, this function always returns NULL. Otherwise, upon success it returns a newly created OctetBuffer whose value is `src`; upon failure it returns NULL.

5.218.4.3 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. 1502), **DDS_OctetBuffer_dup()** (p. 1502)

Parameters

<code>buffer</code>	<< <i>in</i> >> (p. 1013) The buffer to delete.

5.219 SampleProcessor

<<*experimental*>> (p. 1012) <<*extension*>> (p. 1012) Utility to concurrently read and process the data samples received by **DDS_DataReader** (p. 615).

Data Structures

- struct **DDS_SampleHandler**

<<*experimental*>> (p. 1012) <<*extension*>> (p. 1012) <<*interface*>> (p. 1013) Handler called by a sample dispatcher, such as **DDS_SampleProcessor** (p. 1505).

Macros

- #define **DDS_SampleHandler_INITIALIZER**

<<*experimental*>> (p. 1012) <<*extension*>> (p. 1012) Initializer for new **DDS_SampleHandler** (p. 1983).

Typedefs

- typedef struct DDS_SampleProcessorImpl **DDS_SampleProcessor**

<<*extension*>> (p. 1012) <<*interface*>> (p. 1013) A class for reading the samples in **DDS_DataReader** (p. 615)'s cache concurrently. This class allows associating a **DDS_SampleHandler** (p. 1983) with a **DDS_DataReader** (p. 615) and notify it for each sample individually.

Functions

- **DDS_ReturnCode_t DDS_SampleProcessor_attach_reader** (DDS_SampleProcessor *self, DDS_DataReader *reader, const struct DDS_SampleHandler *handler)
*Attaches the specified **DDS_DataReader** (p. 615) with an associated sample handler to this **DDS_SampleProcessor** (p. 1505).*
- **DDS_ReturnCode_t DDS_SampleProcessor_detach_reader** (DDS_SampleProcessor *self, DDS_DataReader *reader)
*Detaches the specified **DDS_DataReader** (p. 615) from this **DDS_SampleProcessor** (p. 1505).*
- **DDS_ReturnCode_t DDS_SampleProcessor_lookup_sample_handler** (DDS_SampleProcessor *self, struct DDS_SampleHandler *handler_out, DDS_DataReader *reader)
*Finds the associated **DDS_SampleHandler** (p. 1983) of the specified attached **DDS_DataReader** (p. 615).*
- **DDS_ReturnCode_t DDS_SampleProcessor_get_datareaders** (DDS_SampleProcessor *self, struct DDS_DataReaderSeq *attached_readers)
*Retrieves the list of attached **DDS_DataReader** (p. 615) (s).*
- **DDS_ReturnCode_t DDS_SampleProcessor_delete** (DDS_SampleProcessor *self)
*Deletes a **DDS_SampleProcessor** (p. 1505).*
- **DDS_SampleProcessor * DDS_SampleProcessor_new** (const struct DDS_AsyncWaitSetProperty_t *aws_property)
*Single-argument constructor that allows creating a **DDS_SampleProcessor** (p. 1505) with the configuration of the underlying **DDS_AsyncWaitSet** (p. 1070).*
- **DDS_SampleProcessor * DDS_SampleProcessor_new_with_aws** (DDS_AsyncWaitSet *aws)
*Constructor that allows specifying an externally created **DDS_AsyncWaitSet** (p. 1070).*

5.219.1 Detailed Description

<<**experimental**>> (p. 1012) <<**extension**>> (p. 1012) Utility to concurrently read and process the data samples received by **DDS_DataReader** (p. 615).

5.219.2 Macro Definition Documentation

5.219.2.1 DDS_SampleHandler_INITIALIZER

```
#define DDS_SampleHandler_INITIALIZER
```

<<**experimental**>> (p. 1012) <<**extension**>> (p. 1012) Initializer for new **DDS_SampleHandler** (p. 1983).

No memory is allocated. New **DDS_SampleHandler** (p. 1983) instances stored in the stack should be initialized with this value before they are passed to any functions.

See also

DDS_SampleHandler (p. 1983)

5.219.3 Typedef Documentation

5.219.3.1 DDS_SampleProcessor

```
typedef struct DDS_SampleProcessorImpl DDS_SampleProcessor
```

<<**extension**>> (p. 1012) <<**interface**>> (p. 1013) A class for reading the samples in **DDS_DataReader** (p. 615)'s cache concurrently. This class allows associating a **DDS_SampleHandler** (p. 1983) with a **DDS_DataReader** (p. 615) and notify it for each sample individually.

A **DDS_SampleProcessor** (p. 1505) relies on an underlying **DDS_AsyncWaitSet** (p. 1070) to read and dispatch the data from the DataReaders. It internally creates a **DDS_ReadCondition** (p. 694) for each attached DataReader, an associates a custom handler that contains state to read samples and notify the corresponding handler .

The SampleProcessor uses this ReadCondition to wait for data and then calls **FooDataReader_take_w_condition** (p. 632) to take all the available data.

Notifications to any handler may be concurrent if the thread pool size set in **DDS_AsyncWaitSetProperty_t::thread↔_pool_size** (p. 1545) is greater than 1. The same or different handlers may be called in parallel. The SampleProcessor cycles through all attached DataReaders dispatching a sample at a time with the next available thread. This mechanism guarantees concurrent dispatching of the samples accross all DataReaders.

The **DDS_SampleProcessor** (p. 1505) internally creates and uses a **DDS_ReadCondition** (p. 694) for each attached **DDS_DataReader** (p. 615) to read the data. It's recommended to perform all the reading of the **DDS_DataReader** (p. 615)'s samples through the **DDS_SampleProcessor** (p. 1505) and avoid reading through other mechanisms in different parts of your applications. Similarly, your application must be careful to not modify the **DDS_ReadCondition** (p. 694) associated with an attached **DDS_DataReader** (p. 615).

In general, avoid or beware of using the following operations in combination with a **DDS_SampleProcessor** (p. 1505):

- read or take operation (and any of its overloads) of a **DDS_DataReader** (p. 615) from any part of your application, including a **DDS_DataReaderListener** (p. 1593) or a handler .
- Delete all the **DDS_DataReader** (p. 615)'s **DDS_ReadCondition** (p. 694) by calling **DDS_DataReader_delete↔_contained_entities** (p. 674)
- Perform actions based on the **DDS_StatusCondition** (p. 1371) with the **DDS_DATA_AVAILABLE_STATUS** (p. 1230) enabled (for example through a **DDS_AsyncWaitSet** (p. 1070)).

On the other hand, you can externally provide the underlying **DDS_SampleProcessor** (p. 1505)'s **DDS_AsyncWaitSet** (p. 1070) if you want to access additional capabilities of the **DDS_AsyncWaitSet** (p. 1070) and use it to attach other **DDS_Condition** (p. 1370) to handle other aspects of your application.

5.219.4 SampleProcessor Thread Safety

Similar to the **DDS_AsyncWaitSet** (p. 1070), the **DDS_SampleProcessor** (p. 1505) provides a thread-safe interface. All the operations of this class can be called concurrently from multiple threads.

Because the **DDS_SampleProcessor** (p. 1505) relies on an **DDS_AsyncWaitSet** (p. 1070) to dispatch the samples the same threading concepts apply. This means that operations on a **DDS_SampleProcessor** (p. 1505) may require synchronizing with the thread pool for safety. **DDS_SampleProcessor** (p. 1505) also relies on the asynchronous completion pattern to effectively interact with the underlying thread pool.

For instance to detach a **DDS_DataReader** (p. 615), the **DDS_SampleProcessor** (p. 1505) 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 on which the **DDS_SampleProcessor** (p. 1505) waits and blocks until it completes.

Due to the concurrent processing nature of operations on a **DDS_SampleProcessor** (p. 1505) as well as the sample handling, it's important to keep in mind the following aspects:

- The handler operation can be called concurrently for each sample. Therefore, handler implementations may need to apply thread synchronization strategies to protect shared resources.
- Calling a **DDS_SampleProcessor** (p. 1505) operation that requires internal synchronization (e.g., **DDS_SampleProcessor_attach_reader** (p. 1506)) from a handler notification will result in a error.

Note that the interface of the **DDS_SampleProcessor** (p. 1505) is similar to that of the **DDS_AsyncWaitSet** (p. 1070) but reduced and simplified. As noted above, if more advanced use and control of the thread pool is required, you can always create the **DDS_AsyncWaitSet** (p. 1070) externally when calling **DDS_SampleProcessor_new_with_aws** (p. 1510).

MT Safety:

Safe.

See also

DDS_AsyncWaitSet (p. 1070)

DDS_Condition (p. 1370)

DDS_AsyncWaitSetProperty_t (p. 1544)

DDS_DataReader (p. 615)

5.219.5 Function Documentation

5.219.5.1 DDS_SampleProcessor_attach_reader()

```
DDS_ReturnCode_t DDS_SampleProcessor_attach_reader (
    DDS_SampleProcessor * self,
    DDS_DataReader * reader,
    const struct DDS_SampleHandler * handler )
```

Attaches the specified **DDS_DataReader** (p.615) with an associated sample handler to this **DDS_SampleProcessor** (p.1505).

This operation will block until the attach request completes. Upon successful return, it is guaranteed that the specified **DDS_DataReader** (p.615) is attached and notifications to the handler may occur.

If this operation is called multiple times for an already attached **DDS_DataReader** (p.615), it will result in no-op and return successfully, ignoring the specified handler. So if the handler is different, it will not be updated.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>reader</i>	<< <i>in</i> >> (p. 1013) DDS_DataReader (p.615) to be attached.
<i>handler</i>	<< <i>in</i> >> (p. 1013) handler to be notified on dispatching of the <i>reader</i> samples.

Returns

One of the **Standard Return Codes** (p.1221)

See also

DDS_SampleProcessor_detach_reader (p.1507)

5.219.5.2 DDS_SampleProcessor_detach_reader()

```
DDS_ReturnCode_t DDS_SampleProcessor_detach_reader (
    DDS_SampleProcessor * self,
    DDS_DataReader * reader )
```

Detaches the specified **DDS_DataReader** (p.615) from this **DDS_SampleProcessor** (p.1505).

Once the **DDS_DataReader** (p.615) is detached, it is guaranteed that the **DDS_SampleProcessor** (p.1505) will no longer process it so it is safe for your application to release any resources associated with the detached **DDS_DataReader** (p.615).

This operation blocks until the detach request completes. Upon successful return, it is guaranteed that the specified **DDS_DataReader** (p.615) is detached.

DDS_DataReader (p.615) may be detached at any time independently of the state of the **DDS_SampleProcessor** (p.1505).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>reader</i>	<< <i>in</i> >> (p. 1013) DDS_DataReader (p. 615) to be detached.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_SampleProcessor_attach_reader (p. 1506)

5.219.5.3 DDS_SampleProcessor_lookup_sample_handler()

```
DDS_ReturnCode_t DDS_SampleProcessor_lookup_sample_handler (
    DDS_SampleProcessor * self,
    struct DDS_SampleHandler * handler_out,
    DDS_DataReader * reader )
```

Finds the associated **DDS_SampleHandler** (p. 1983) of the specified attached **DDS_DataReader** (p. 615).

The operation will fail with **DDS_RETCODE_PRECONDITION_NOT_MET** (p. 1222). if the specified reader is not currently attached.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>handler_out</i>	<< <i>inout</i> >> (p. 1013) a DDS_SampleHandler (p. 1983) object that contains the found associated handler.
<i>reader</i>	DDS_DataReader (p. 615) The attached reader for which the associated DDS_SampleHandler (p. 1983) is retrieved.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_SampleProcessor_attach_reader (p. 1506)

DDS_SampleProcessor_detach_reader (p. 1507)

5.219.5.4 DDS_SampleProcessor_get_datareaders()

```
DDS_ReturnCode_t DDS_SampleProcessor_get_datareaders (
    DDS_SampleProcessor * self,
    struct DDS_DataReaderSeq * attached_readers )
```

Retrieves the list of attached **DDS_DataReader** (p. 615) (s).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>attached_readers</i>	<< <i>inout</i> >> (p. 1013) a DDS_DataReaderSeq (p. 1637) object where the list of attached readers will be returned.

Returns

One of the **Standard Return Codes** (p. 1221)

See also

DDS_SampleProcessor_attach_reader (p. 1506)

DDS_SampleProcessor_detach_reader (p. 1507)

5.219.5.5 DDS_SampleProcessor_delete()

```
DDS_ReturnCode_t DDS_SampleProcessor_delete (
    DDS_SampleProcessor * self )
```

Deletes a **DDS_SampleProcessor** (p. 1505).

This will delete the underlying **DDS_AsyncWaitSet** (p. 1070) only if it owns it. That is, if the **DDS_AsyncWaitSet** (p. 1070) was not provided externally.

The deletion will fail if there are outstanding **DDS_AsyncWaitSetCompletionToken** (p. 1073) that have not been deleted.

Any outstanding **DDS_SampleProcessor** (p. 1505) must be deleted before finalizing the **DDS_DomainParticipantFactory** (p. 32). Otherwise undefined behavior may occur.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

See also

DDS_SampleProcessor_new (p. 1510)

5.219.5.6 DDS_SampleProcessor_new()

```
DDS_SampleProcessor * DDS_SampleProcessor_new (
    const struct DDS_AsyncWaitSetProperty_t * aws_property )
```

Single-argument constructor that allows creating a **DDS_SampleProcessor** (p. 1505) with the configuration of the underlying **DDS_AsyncWaitSet** (p. 1070).

You can provide **DDS_ASYNC_WAITSET_PROPERTY_DEFAULT** (p. 1090) as *property* to create the underlying **DDS_AsyncWaitSet** (p. 1070) with default behavior.

This constructor creates **DDS_AsyncWaitSet** (p. 1070) with no listener installed and will call **DDS_AsyncWaitSet_start** (p. 1082) right after its creation.

Parameters

<i>aws_property</i>	<< <i>in</i> >> (p. 1013) configuration of the underlying DDS_AsyncWaitSet (p. 1070).

Returns

A new **DDS_SampleProcessor** (p. 1505) or NULL if one could not be allocated.

See also

DDS_SampleProcessor_new_with_aws (p. 1510)

5.219.5.7 DDS_SampleProcessor_new_with_aws()

```
DDS_SampleProcessor * DDS_SampleProcessor_new_with_aws (
    DDS_AsyncWaitSet * aws )
```

Constructor that allows specifying an externally created **DDS_AsyncWaitSet** (p. 1070).

Creates a new **DDS_SampleProcessor** (p. 1505) with the specified **DDS_AsyncWaitSet** (p. 1070).

This constructor flavor decouples the lifecycle of the **DDS_SampleProcessor** (p. 1505) from the **DDS_AsyncWaitSet** (p. 1070). It allows the application to have more control on the **DDS_AsyncWaitSet** (p. 1070), such as to install an **DDS_AsyncWaitSetListener** (p. 1543), a custom **DDS_ThreadFactory** (p. 2027), and controlling when it starts or stops.

Note that this constructor will not call **DDS_AsyncWaitSet_start** (p. 1082), so it's the caller responsibility to start and stop it. You can provide the external **DDS_AsyncWaitSet_start** (p. 1082) in either started or stopped state.

Parameters

<i>aws</i>	<< <i>in</i> >> (p. 1013) the externally created DDS_AsyncWaitSet (p. 1070).

Returns

A new **DDS_SampleProcessor** (p. 1505) or NULL if one could not be allocated.

See also

DDS_SampleProcessor_new (p. 1510)

5.220 Sequence Support

The **FooSeq** (p. 2113) interface allows you to work with variable-length collections of homogeneous data.

Modules

- **Built-in Sequences**

Defines sequences of primitive data type. .

Data Structures

- struct **FooSeq**

<<**interface**>> (p. 1013) <<**generic**>> (p. 1013) *A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as **Foo** (p. 2109).*

Macros

- #define **DDS_SEQUENCE_INITIALIZER**

An initializer for new sequence instances.

Functions

- **DDS_Boolean FooSeq_initialize** (struct **FooSeq** *self)
Initialize sequence instances.
- **DDS_Long FooSeq_get_maximum** (const struct **FooSeq** *self)
Get the current maximum number of elements that can be stored in this sequence.
- **DDS_Boolean FooSeq_set_maximum** (struct **FooSeq** *self, **DDS_Long** new_max)
Resize this sequence to a new desired maximum.
- **DDS_Long FooSeq_get_length** (const struct **FooSeq** *self)
Get the logical length of this sequence.
- **DDS_Boolean FooSeq_set_length** (struct **FooSeq** *self, **DDS_Long** new_length)
Set the sequence to the desired length, and resize the sequence if necessary.
- **DDS_Boolean FooSeq_ensure_length** (struct **FooSeq** *self, **DDS_Long** length, **DDS_Long** max)
Set the sequence to the desired length, and resize the sequence if necessary.
- **Foo FooSeq_get** (const struct **FooSeq** *self, **DDS_Long** i)
Get the i-th element for a const sequence.
- **Foo * FooSeq_get_reference** (const struct **FooSeq** *self, **DDS_Long** i)
Get the pointer to the i-th element of this sequence.
- **DDS_Boolean FooSeq_copy_no_alloc** (struct **FooSeq** *self, const struct **FooSeq** *src_seq)
Copy elements from another sequence, only if the destination sequence has enough capacity.
- **FooSeq * FooSeq_copy** (struct **FooSeq** *self, const struct **FooSeq** *src_seq)
Copy elements from another sequence, resizing the sequence if necessary.
- **DDS_Boolean FooSeq_from_array** (struct **FooSeq** *self, 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.
- **DDS_Boolean FooSeq_to_array** (struct **FooSeq** *self, **Foo** array[], **DDS_Long** length)
Copy elements to an array of elements. The original contents of the array (if any) are replaced.
- **DDS_Boolean FooSeq_loan_contiguous** (struct **FooSeq** *self, **Foo** *buffer, **DDS_Long** new_length, **DDS_Long** new_max)
Loan a contiguous buffer to this sequence.
- **DDS_Boolean FooSeq_loan_discontiguous** (struct **FooSeq** *self, **Foo** **buffer, **DDS_Long** new_length, **DDS_Long** new_max)
Loan a discontiguous buffer to this sequence.
- **DDS_Boolean FooSeq_unloan** (struct **FooSeq** *self)
Return the loaned buffer in the sequence and set the maximum to 0.
- **Foo * FooSeq_get_contiguous_buffer** (const struct **FooSeq** *self)
Return the contiguous buffer of the sequence.
- **Foo ** FooSeq_get_discontiguous_buffer** (const struct **FooSeq** *self)
Return the discontiguous buffer of the sequence.
- **DDS_Boolean FooSeq_has_ownership** (const struct **FooSeq** *self)
Return the value of the owned flag.
- **DDS_Boolean FooSeq_finalize** (struct **FooSeq** *self)
Deallocate this sequence's buffer.

5.220.1 Detailed Description

The **FooSeq** (p. 2113) 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. 718) 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.

5.220.2 Macro Definition Documentation

5.220.2.1 DDS_SEQUENCE_INITIALIZER

```
#define DDS_SEQUENCE_INITIALIZER
```

An initializer for new sequence instances.

This constant will initialize a new sequence to a valid empty state. C language users should assign it to uninitialized sequence instances before using them, at the time they are declared, or use **FooSeq_initialize** (p. 1513) as an alternative function to initialize new sequences.

Examples

HelloWorld_subscriber.c.

5.220.3 Function Documentation

5.220.3.1 FooSeq_initialize()

```
DDS_Boolean FooSeq_initialize (
    struct FooSeq * self )
```

Initialize sequence instances.

Use this function to initialize new sequences to a valid empty state. C users should initialize sequences before using them.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

See also

DDS_SEQUENCE_INITIALIZER (p. 1513)

5.220.3.2 FooSeq_get_maximum()

```
DDS_Long FooSeq_get_maximum (
    const struct FooSeq * self )
```

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_set_maximum** (p. 1514) operation.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

the current maximum of the sequence.

See also

FooSeq_get_length (p. 1515)

5.220.3.3 FooSeq_set_maximum()

```
DDS_Boolean FooSeq_set_maximum (
    struct FooSeq * self,
    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. 1201)
 new_max <= maximum size for IDL bounded sequences.

Postcondition

owned == **DDS_BOOLEAN_TRUE** (p. 1201)
 length == MINIMUM(original length, new_max)

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>new_max</i>	Must be >= 0.

Returns

DDS_BOOLEAN_TRUE (p. 1201) on success, **DDS_BOOLEAN_FALSE** (p. 1201) if the preconditions are not met. In that case the sequence is not modified.

5.220.3.4 FooSeq_get_length()

```
DDS_Long FooSeq_get_length (
    const struct FooSeq * self )
```

Get the logical length of this sequence.

Get the length that was last set, or zero if the length has never been set.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

the length of the sequence

5.220.3.5 FooSeq_set_length()

```
DDS_Boolean FooSeq_set_length (
    struct FooSeq * self,
    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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>new_length</i>	the new desired length. This value must be non-negative. Must be ≥ 0 .

Returns

DDS_BOOLEAN_TRUE (p. 1201) on success or **DDS_BOOLEAN_FALSE** (p. 1201) on failure

5.220.3.6 FooSeq_ensure_length()

```
DDS_Boolean FooSeq_ensure_length (
    struct FooSeq * self,
    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. 1201) if sequence needs to be resized
```

Postcondition

```
length == length
maximum == max if resized
```

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>length</i>	<< <i>in</i> >> (p. 1013) The new length that should be set. Must be ≥ 0 .
<i>max</i>	<< <i>in</i> >> (p. 1013) If sequence need to be resized, this is the maximum that should be set. <code>max</code> $\geq$ length

Returns

DDS_BOOLEAN_TRUE (p. 1201) on success, **DDS_BOOLEAN_FALSE** (p. 1201) if the preconditions are not met. In that case the sequence is not modified.

5.220.3.7 FooSeq_get()

```

Foo FooSeq_get (
    const struct FooSeq * self,
    DDS_Long i )

```

Get the *i*-th element for a `const` sequence.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>i</i>	index of element to access, must be ≥ 0 and less than FooSeq_get_length (p. 1515)

Returns

the *i*-th element

5.220.3.8 FooSeq_get_reference()

```

Foo * FooSeq_get_reference (
    const struct FooSeq * self,
    DDS_Long i )

```

Get the pointer to the *i*-th element of this sequence.

This operation can used to modify the elements of the sequence in place.

```

struct DDS_LongSeq my_seq = DDS_SEQUENCE_INITIALIZER;

DDS_Long* first_element = NULL;

DDS_LongSeq_ensure_length(1, 1);

first_element = DDS_LongSeq_get_reference(&my_seq, 0);

*first_element = 5;

```

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>i</i>	index of element to access, must be ≥ 0 and less than <code>FooSeq_get_length</code> (p. 1515)

Returns

a pointer to the `i`-th element

5.220.3.9 `FooSeq_copy_no_alloc()`

```
DDS_Boolean FooSeq_copy_no_alloc (
    struct FooSeq * self,
    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. 1201)
```

Postcondition

```
this::length == src_seq::length
this[i] == src_seq[i] for  $0 \leq i < \text{target\_seq::length}$ 
this::owned == DDS_BOOLEAN_TRUE (p. 1201)
```

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>src_seq</i>	<< <i>in</i> >> (p. 1013) the sequence from which to copy

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the sequence was successfully copied; **DDS_BOOLEAN_FALSE** (p. 1201) 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_copy (p. 1519)

5.220.3.10 FooSeq_copy()

```
FooSeq * FooSeq_copy (
    struct FooSeq * self,
    const struct FooSeq * src_seq )
```

Copy elements from another sequence, resizing the sequence if necessary.

This function invokes **FooSeq_copy_no_alloc** (p. 1518) after ensuring that the sequence has enough capacity to hold the elements to be copied.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>src_seq</i>	<< <i>in</i> >> (p. 1013) the sequence from which to copy

Returns

self, this sequence

See also

FooSeq_copy_no_alloc (p. 1518)

5.220.3.11 FooSeq_from_array()

```
DDS_Boolean FooSeq_from_array (
    struct FooSeq * self,
    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. 1201)

Postcondition

`this::length == length`

`this[i] == array[i]` for $0 \leq i < \text{length}$

`this::owned == DDS_BOOLEAN_TRUE` (p. 1201)

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>array</i>	<< <i>in</i> >> (p. 1013) The array of elements to be copy elements from
<i>length</i>	<< <i>in</i> >> (p. 1013) The length of the array.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the array was successfully copied; **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

Note

If the pre-conditions are not met, the function will print a message to stdout and leave this sequence unchanged.

5.220.3.12 FooSeq_to_array()

```
DDS_Boolean FooSeq_to_array (
    struct FooSeq * self,
    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

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>array</i>	<< <i>in</i> >> (p. 1013) The array of elements to be filled with elements from this sequence
<i>length</i>	<< <i>in</i> >> (p. 1013) The number of elements to be copied.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the elements of the sequence were successfully copied; **DDS_BOOLEAN_FALSE** (p. 1201) otherwise.

5.220.3.13 FooSeq_loan_contiguous()

```
DDS_Boolean FooSeq_loan_contiguous (
    struct FooSeq * self,
    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. 1201) 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 function 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_set_maximum** (p. 1514).

Once you have loaned a buffer to a sequence, make sure that you don't free it before calling **FooSeq_unloan** (p. 1522): the next time you access the sequence, you will be accessing freed memory!

You can use this function to wrap stack memory with a sequence interface, thereby avoiding dynamic memory allocation. Create a **FooSeq** (p. 2113) and an array of type **Foo** (p. 2109) and then loan the array to the sequence:

```
struct ::Foo fooArray[10];

struct ::FooSeq fooSeq = ::DDS_SEQUENCE_INITIALIZER;

::FooSeq_loan_contiguous(&fooSeq, 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 Connexx will return a sequence to you that has a loan; those cases are documented as such. For example, if you call **FooDataReader_read** (p. 625) or **FooDataReader_take** (p. 626) and pass in sequences with no loan and no memory allocated, RTI Connexx will loan memory to your sequences which must be unloaned with **FooDataReader_return_loan** (p. 646). See the documentation of those functions for more information.

Precondition

FooSeq_get_maximum (p. 1514) == 0; i.e. the sequence has no memory allocated to it.

FooSeq_has_ownership (p. 1524) == **DDS_BOOLEAN_TRUE** (p. 1201); 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. 1524) == **DDS_BOOLEAN_FALSE** (p. 1201)

FooSeq_get_length (p. 1515) == `new_length`

FooSeq_get_maximum (p. 1514) == `new_max`

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>buffer</i>	The new buffer that the sequence will use. Must point to enough memory to hold <i>new_max</i> elements of type Foo (p. 2109). It may be NULL if <i>new_max</i> == 0.
<i>new_length</i>	The desired new length for the sequence. It must be the case that that 0 <= <i>new_length</i> <= <i>new_max</i> .
<i>new_max</i>	The allocated number of elements that could fit in the loaned buffer.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if *buffer* is successfully loaned to this sequence or **DDS_BOOLEAN_FALSE** (p. 1201) otherwise. Failure only occurs due to failing to meet the pre-conditions. Upon failure the sequence remains unmodified.

See also

FooSeq_unloan (p. 1522) , **FooSeq_loan_discontiguous** (p. 1522)

5.220.3.14 FooSeq_loan_discontiguous()

```
DDS_Boolean FooSeq_loan_discontiguous (
    struct FooSeq * self,
    Foo ** buffer,
    DDS_Long new_length,
    DDS_Long new_max )
```

Loan a discontiguous buffer to this sequence.

This function is exactly like **FooSeq_loan_contiguous** (p. 1521) except that the buffer loaned is an array of **Foo** (p. 2109) pointers, not an array of **Foo** (p. 2109).

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.
<i>buffer</i>	The new buffer that the sequence will use. Must point to enough memory to hold <i>new_max</i> elements of type Foo* . It may be NULL if <i>new_max</i> == 0.
<i>new_length</i>	The desired new length for the sequence. It must be the case that that 0 <= <i>new_length</i> <= <i>new_max</i> .
<i>new_max</i>	The allocated number of elements that could fit in the loaned buffer.

See also

FooSeq_unloan (p. 1522), **FooSeq_loan_contiguous** (p. 1521)

5.220.3.15 FooSeq_unloan()

```
DDS_Boolean FooSeq_unloan (
    struct FooSeq * self )
```

Return the loaned buffer in the sequence and set the maximum to 0.

This function 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. 1201)

Postcondition

owned == **DDS_BOOLEAN_TRUE** (p. 1201)

maximum == 0

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if the preconditions were met. Otherwise **DDS_BOOLEAN_FALSE** (p. 1201). 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. 1521), **FooSeq_loan_discontiguous** (p. 1522), **FooSeq_set_maximum** (p. 1514)

5.220.3.16 FooSeq_get_contiguous_buffer()

```
Foo * FooSeq_get_contiguous_buffer (
    const struct FooSeq * self )
```

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_get_length** (p. 1515) of the sequence are valid.

This function provides almost no encapsulation of the sequence's underlying implementation. Certain operations, such as **FooSeq_set_maximum** (p. 1514), may render the buffer invalid. In light of these caveats, this operation should be used with care.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

buffer that stores contiguous elements in sequence.

5.220.3.17 FooSeq_get_discontiguous_buffer()

```
Foo ** FooSeq_get_discontiguous_buffer (
    const struct FooSeq * self )
```

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_get_length** (p. 1515) of the sequence are valid.

The same caveats apply to this function as to **FooSeq_get_contiguous_buffer** (p. 1523).

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 function, do not store any NULL values into it, as accessing those values will result in a segmentation fault.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

buffer that stores discontiguous elements in sequence.

5.220.3.18 FooSeq_has_ownership()

```
DDS_Boolean FooSeq_has_ownership (
    const struct FooSeq * self )
```

Return the value of the owned flag.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Returns

DDS_BOOLEAN_TRUE (p. 1201) if sequence owns the underlying buffer, or **DDS_BOOLEAN_FALSE** (p. 1201) if it has an outstanding loan.

5.220.3.19 FooSeq_finalize()

```
DDS_Boolean FooSeq_finalize (
    struct FooSeq * self )
```

Deallocate this sequence's buffer.

Note that this function deletes this sequence's *buffer*, not this sequence itself. To avoid memory leaks, it should be called even on sequences declared on the stack. And after it has been called on a sequence located in dynamic memory, it is still necessary to call `free()`.

Precondition

(owned == **DDS_BOOLEAN_TRUE** (p. 1201)). If this precondition is not met, no memory will be freed and an error will be logged.

Parameters

<i>self</i>	<< <i>in</i> >> (p. 1013) Cannot be NULL.

Postcondition

maximum == 0 and the underlying buffer is freed.

See also

FooSeq_get_maximum (p. 1514), **FooSeq_unloan** (p. 1522)

5.221 String Support

<<**extension**>> (p. 1012) 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.

5.221.1 Detailed Description

<<**extension**>> (p. 1012) String creation, cloning, assignment, and deletion.

The functions in this class ensure consistent cross-platform implementations for string creation (**DDS_String_alloc()** (p. 1528)), deletion (**DDS_String_free()** (p. 1529)), and cloning (**DDS_String_dup()** (p. 1528)) that preserve the mutable value type semantics. These are to be viewed as functions that define a `string` class whose data is represented by a `'char*'`.

5.221.2 String Conventions

The following conventions govern the memory management of strings in RTI Connex.

- 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 functions.
 - 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. 2005) is a value type that contains strings; it owns the memory for the contained strings. When a **DDS_StringSeq** (p. 2005) 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 functions.

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 Connex must therefore make some assumptions when a user requests that a string be copied into. The following rules apply when RTI Connex is copying into a string or string sequence:

- If the 'char*' is NULL, RTI Connex 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. 1527) below).*
- If the 'char*' is not NULL, RTI Connex will assume that you are managing the string's memory yourself and have allocated enough memory to store the string to be copied. *RTI Connex 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 Connex; you are likely to crash.*

5.221.3 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. 1528), **DDS_String_dup()** (p. 1528), with matching occurrences of **DDS_String_free()** (p. 1529) 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.1922) and **DDS_MultiChannelQosPolicy::filter_name** (p. 1858) 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. 2005)

5.221.4 Function Documentation

5.221.4.1 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 function must be deleted using **DDS_String_free()** (p. 1529).

This function will allocate enough memory to hold a string of `length` characters, **plus** one additional byte to hold the NULL terminating character.

Parameters

<i>length</i>	<< <i>in</i> >> (p. 1013) Capacity of the string.

Returns

A newly created non-NULL string upon success or NULL upon failure.

Examples

HelloWorld.c.

5.221.4.2 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 function must be deleted using **DDS_String_free()** (p. 1529)

Parameters

<i>str</i>	<< <i>in</i> >> (p. 1013) The string to duplicate.

Returns

If `string == NULL`, this function always returns `NULL`. Otherwise, upon success it returns a newly created string whose value is `string`; upon failure it returns `NULL`.

5.221.4.3 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 function must be deleted using **DDS_String_free()** (p. 1529).

This function is most commonly used when manipulating string sequences, **DDS_StringSeq** (p. 2005).

Precondition

`string_ptr` be a non-NULL pointer. `*string_ptr` must be either `NULL`, or a string created using **DDS_String_alloc()** (p. 1528) or **DDS_String_dup()** (p. 1528), or **DDS_String_replace()** (p. 1529).

Parameters

<i>string_ptr</i>	<< <i>inout</i> >> (p. 1013) Pointer to the string to replace.
<i>new_value</i>	<< <i>in</i> >> (p. 1013) The value of the replacement string.

Returns

If `new_value = NULL`, this function 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.

5.221.4.4 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. 1528), **DDS_String_←dup()** (p. 1528)

Parameters

<i>str</i>	<< <i>in</i> >> (p. 1013) The string to delete.

Examples

HelloWorld.c.

5.221.4.5 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 function must be deleted using **DDS_Wstring_free()** (p. 1532)

This function will allocate enough memory to hold a string of `length` characters, **plus** one additional wide character to hold the NULL terminator.

Parameters

<i>length</i>	<< <i>in</i> >> (p. 1013) Capacity of the string.

Returns

A newly created non-NULL string upon success or NULL upon failure.

5.221.4.6 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

<i>str</i>	<< <i>in</i> >> (p. 1013) A non-NULL string.

Returns

The number of wide characters in the string.

5.221.4.7 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

<i>dst</i>	
<i>src</i>	

Returns

dst

5.221.4.8 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

<i>dst</i>	<< <i>in</i> >> (p. 1013) A non-NULL string to be overwritten by <i>src</i> .
<i>src</i>	<< <i>in</i> >> (p. 1013) A non-NULL string to be copied over <i>dst</i>

Returns

`dst`

5.221.4.9 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 function must be deleted using **DDS_Wstring_free()** (p. 1532).

Parameters

<i>str</i>	<< <i>in</i> >> (p. 1013) The string to duplicate.

Returns

If `string == NULL`, this function always returns `NULL`. Otherwise, upon success it returns a newly created string whose value is `string`; upon failure it returns `NULL`.

5.221.4.10 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 function must be deleted using **DDS_Wstring_free()** (p. 1532)

Parameters

<i>str</i>	<< <i>in</i> >> (p. 1013) The string to duplicate.

Returns

If `string == NULL`, this function always returns `NULL`. Otherwise, upon success it returns a newly created string whose value is `string`; upon failure it returns `NULL`.

5.221.4.11 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. 1530), **DDS_Wstring_↵
dup()** (p. 1532), or **DDS_Wstring_replace()**

Parameters

<i>str</i>	<< <i>in</i> >> (p. 1013) The string to delete.

Chapter 6

Data Structure Documentation

6.1 DDS_AcknowledgmentInfo Struct Reference

Information about an application-level acknowledged sample.

Data Fields

- **DDS_InstanceHandle_t subscription_handle**
*Subscription handle of the acknowledging **DDS_DataReader** (p. 615).*
- 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.

6.1.1 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.

6.1.2 Field Documentation

6.1.2.1 subscription_handle

DDS_InstanceHandle_t DDS_AcknowledgmentInfo::subscription_handle

Subscription handle of the acknowledging **DDS_DataReader** (p. 615).

6.1.2.2 sample_identity

struct DDS_SampleIdentity_t DDS_AcknowledgmentInfo::sample_identity

Identity of the sample being acknowledged.

See also

DDS_SampleIdentity_t (p. 1984)

6.1.2.3 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. 1958) has not yet elapsed for the acknowledgment's response data.

The flag is false when that duration has elapsed for the response data.

6.1.2.4 response_data

struct DDS_AckResponseData_t DDS_AcknowledgmentInfo::response_data

User data payload of application-level acknowledgment message.

Response data set by **DDS_DataReader** (p. 615) when sample was acknowledged.

6.2 DDS_AckResponseData_t Struct Reference

Data payload of an application-level acknowledgment.

Data Fields

- struct **DDS_OctetSeq** **value**
a sequence of octets

6.2.1 Detailed Description

Data payload of an application-level acknowledgment.

6.2.2 Field Documentation

6.2.2.1 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. 1633)],

6.3 DDS_AllocationSettings_t Struct Reference

Resource allocation settings.

Data Fields

- **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.

6.3.1 Detailed Description

Resource allocation settings.

QoS:

DDS_DomainParticipantResourceLimitsQosPolicy (p. 1731)

6.3.2 Field Documentation

6.3.2.1 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, (or = max_count only if increment_count == 0)

6.3.2.2 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. 1326), > initial_count (or = initial_count only if increment_count == 0)

6.3.2.3 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)

6.4 DDS_AnnotationParameterValue Struct Reference

Annotation parameter value.

6.4.1 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.

6.5 DDS_AsyncronousPublisherQosPolicy Struct Reference

<<**extension**>> (p. 795) Configures the mechanism that sends user data in an external middleware thread.

Data Fields

- **DDS_Boolean disable_asynchronous_write**
[DEPRECATED] (p. 965) <<**extension**>> (p. 795) Disable asynchronous publishing.
- struct **DDS_ThreadSettings_t thread**
<<**extension**>> (p. 795) Settings of the publishing thread.
- **DDS_Boolean disable_asynchronous_batch**
[DEPRECATED] (p. 965) <<**extension**>> (p. 795) Disable asynchronous batch flushing.
- struct **DDS_ThreadSettings_t asynchronous_batch_thread**
<<**extension**>> (p. 795) Settings of the batch flushing thread.
- **DDS_Boolean disable_topic_query_publication**
[DEPRECATED] (p. 965) <<**extension**>> (p. 795) Disable topic query publication.
- struct **DDS_ThreadSettings_t topic_query_publication_thread**
<<**extension**>> (p. 795) Settings of the **DDS_TopicQuery** (p. 707) publication thread.

6.5.1 Detailed Description

<<**extension**>> (p. 795) Configures the mechanism that sends user data in an external middleware thread.

Specifies the asynchronous publishing and asynchronous batch flushing settings of the **DDS_Publisher** (p. 443) instances.

The QoS policy specifies whether asynchronous publishing and asynchronous batch flushing are enabled for the **DDS_↵DataWriter** (p. 485) entities belonging to this **DDS_Publisher** (p. 443). 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 **DDS_TopicQuery** (p. 707) publication thread. The publisher will spawn this thread only if one or more DataWriters enable TopicQueries.

See also

DDS_BatchQosPolicy (p. 1552).

DDS_PublishModeQosPolicy (p. 1920).

Entity:

DDS_Publisher (p. 443)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

6.5.2 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. 1319) **DDS_↔_DataWriter** (p. 485) instances that belong to this publisher and handles their data transmission chores.

The asynchronous *batch flushing* thread is shared by all **DDS_DataWriter** (p. 485) 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 **DDS_DataWriter** (p. 485) entities, the instances must belong to different **DDS_Publisher** (p. 443) instances.

A **DDS_Publisher** (p. 443) must have asynchronous publishing enabled for its **DDS_DataWriter** (p. 485) instances to write asynchronously.

A **DDS_Publisher** (p. 443) must have asynchronous batch flushing enabled in order to flush the batches of its **DDS_↔_DataWriter** (p. 485) instances asynchronously. However, no asynchronous batch flushing thread will be started until the first **DDS_DataWriter** (p. 485) instance with batching enabled is created from this **DDS_Publisher** (p. 443).

6.5.3 Field Documentation

6.5.3.1 disable_asynchronous_write

DDS_Boolean DDS_AsynchronousPublisherQosPolicy::disable_asynchronous_write

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Disable asynchronous publishing.

If set to **DDS_BOOLEAN_TRUE** (p. 1201), any **DDS_DataWriter** (p. 485) created with **DDS_ASYNCHRONOUS_↔_PUBLISH_MODE_QOS** (p. 1319) will fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222).

[default] **DDS_BOOLEAN_FALSE** (p. 1201)

Categories

Immutable (p. 914), **Flow Control** (p. 863), **Concurrency** (p. 839)

6.5.3.2 thread

```
struct DDS_ThreadSettings_t DDS_AsyncronousPublisherQosPolicy::thread
```

<<**extension**>> (p. 795) Settings of the publishing thread.

There is only one asynchronous publishing thread per **DDS_Publisher** (p. 443).

[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. 1236)

Categories

Immutable (p. 914), **Concurrency** (p. 839)

6.5.3.3 disable_asynchronous_batch

```
DDS_Boolean DDS_AsyncronousPublisherQosPolicy::disable_asynchronous_batch
```

[DEPRECATED] (p. 965) <<**extension**>> (p. 795) Disable asynchronous batch flushing.

If set to **DDS_BOOLEAN_TRUE** (p. 1201), any **DDS_DataWriter** (p. 485) created with batching enabled will fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222).

If **DDS_BatchQosPolicy::max_flush_delay** (p. 1554) is different than **DDS_DURATION_INFINITE** (p. 1208), **DDS_↔ AsyncronousPublisherQosPolicy::disable_asynchronous_batch** (p. 1541) must be set **DDS_BOOLEAN_FALSE** (p. 1201).

[default] **DDS_BOOLEAN_FALSE** (p. 1201)

Categories

Immutable (p. 914), **Concurrency** (p. 839), **Batching** (p. 838)

6.5.3.4 asynchronous_batch_thread

```
struct DDS_ThreadSettings_t DDS_AynchronousPublisherQosPolicy::asynchronous_batch_thread
```

<<**extension**>> (p. 795) Settings of the batch flushing thread.

There is only one asynchronous batch flushing thread per **DDS_Publisher** (p. 443).

[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. 1236)

Categories

Immutable (p. 914), **Concurrency** (p. 839), **Batching** (p. 838)

6.5.3.5 disable_topic_query_publication

```
DDS_Boolean DDS_AynchronousPublisherQosPolicy::disable_topic_query_publication
```

[DEPRECATED] (p. 965) <<**extension**>> (p. 795) Disable topic query publication.

If set to **DDS_BOOLEAN_TRUE** (p. 1201), any **DDS_DataWriter** (p. 485) created with **DDS_TopicQueryDispatch**↵
QosPolicy::enable (p. 2048) set to **DDS_BOOLEAN_TRUE** (p. 1201) will fail with **DDS_RETCODE_INCONSISTENT**↵
_POLICY (p. 1222).

[default] **DDS_BOOLEAN_FALSE** (p. 1201)

Categories

Immutable (p. 914), **Concurrency** (p. 839), **Topic Queries** (p. 911)

6.5.3.6 topic_query_publication_thread

```
struct DDS_ThreadSettings_t DDS_AsyncronousPublisherQosPolicy::topic_query_publication_thread
```

<<**extension**>> (p. 795) Settings of the **DDS_TopicQuery** (p. 707) publication thread.

There is only one TopicQuery publication thread per **DDS_Publisher** (p. 443). This thread will exist as long as one or more **DDS_DataWriter** (p. 485) enables TopicQueries (via **DDS_DataWriterQos::topic_query_dispatch** (p. 1673)).

[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. 1236)

Categories

Immutable (p. 914), **Concurrency** (p. 839), **Topic Queries** (p. 911)

6.6 DDS_AsyncWaitSetListener Struct Reference

<<**interface**>> (p. 1013) Listener for receiving event notifications related to the thread pool of the **DDS_AsyncWaitSet** (p. 1070).

Data Fields

- void * **listener_data**
A place for listener implementors to keep a pointer to data that may be needed by their listener.
- **DDS_AsyncWaitSetListener_OnThreadSpawnedCallback** **on_thread_spawned**
Handles the spawning of each thread comprising the thread pool of the **DDS_AsyncWaitSet** (p. 1070).
- **DDS_AsyncWaitSetListener_OnThreadDeletedCallback** **on_thread_deleted**
Handles the deletion of each thread comprising the thread pool of the **DDS_AsyncWaitSet** (p. 1070).
- **DDS_AsyncWaitSetListener_OnWaitTimeoutCallback** **on_wait_timeout**
Handles the wait timeout generated by the leader thread of the **DDS_AsyncWaitSet** (p. 1070).

6.6.1 Detailed Description

<<*interface*>> (p. 1013) Listener for receiving event notifications related to the thread pool of the **DDS_AsyncWaitSet** (p. 1070).

6.6.2 Field Documentation

6.6.2.1 listener_data

```
void* DDS_AsyncWaitSetListener::listener_data
```

A place for listener implementors to keep a pointer to data that may be needed by their listener.

6.6.2.2 on_thread_spawned

```
DDS_AsyncWaitSetListener_OnThreadSpawnedCallback DDS_AsyncWaitSetListener::on_thread_spawned
```

Handles the spawning of each thread comprising the thread pool of the **DDS_AsyncWaitSet** (p. 1070).

6.6.2.3 on_thread_deleted

```
DDS_AsyncWaitSetListener_OnThreadDeletedCallback DDS_AsyncWaitSetListener::on_thread_deleted
```

Handles the deletion of each thread comprising the thread pool of the **DDS_AsyncWaitSet** (p. 1070).

6.6.2.4 on_wait_timeout

```
DDS_AsyncWaitSetListener_OnWaitTimeoutCallback DDS_AsyncWaitSetListener::on_wait_timeout
```

Handles the wait timeout generated by the leader thread of the **DDS_AsyncWaitSet** (p. 1070).

6.7 DDS_AsyncWaitSetProperty_t Struct Reference

Specifies the **DDS_AsyncWaitSet** (p. 1070) behavior.

Data Fields

- struct **DDS_WaitSetProperty_t** **waitset_property**
*Specifies the behavior of the asynchronous wait behavior, which is equivalent to the wait mechanism of a **DDS_WaitSet** (p. 1371).*
- **DDS_UnsignedLong** **thread_pool_size**
*Number of threads that comprise the thread pool of the **DDS_AsyncWaitSet** (p. 1070).*
- struct **DDS_ThreadSettings_t** **thread_settings**
***DDS_ThreadSettings_t** (p. 2029) for each thread comprising the thread pool of the **DDS_AsyncWaitSet** (p. 1070).*
- char * **thread_name_prefix**
*Prefix used to compose the name of each thread that comprises the thread pool of the **DDS_AsyncWaitSet** (p. 1070).*
- struct **DDS_Duration_t** **wait_timeout**
Asynchronous wait timeout.
- **DDS_Long** **level**
*Specifies the level of an **DDS_AsyncWaitSet** (p. 1070).*

6.7.1 Detailed Description

Specifies the **DDS_AsyncWaitSet** (p. 1070) behavior.

This property allows configuring the behavior of the asynchronous wait and the **DDS_Condition** (p. 1370) dispatch, as well as the parameters of the thread pool.

See also

DDS_WaitSetProperty_t (p. 2091)

DDS_ThreadSettings_t (p. 2029)

6.7.2 Field Documentation

6.7.2.1 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 **DDS_WaitSet** (p. 1371).

[default] **DDS_WaitSetProperty_t_INITIALIZER** (p. 1369)

See also

DDS_WaitSetProperty_t (p. 2091)

6.7.2.2 thread_pool_size

```
DDS_UnsignedLong DDS_AsyncWaitSetProperty_t::thread_pool_size
```

Number of threads that comprise the thread pool of the **DDS_AsyncWaitSet** (p. 1070).

Size must be equal to or greater than one.

[default] 1

6.7.2.3 thread_settings

```
struct DDS_ThreadSettings_t DDS_AsyncWaitSetProperty_t::thread_settings
```

DDS_ThreadSettings_t (p. 2029) for each thread comprising the thread pool of the **DDS_AsyncWaitSet** (p. 1070).

Each thread within the pool is created with the same settings.

[default] Default thread settings values.

See also

DDS_ThreadSettings_t (p. 2029)

6.7.2.4 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 **DDS_AsyncWaitSet** (p. 1070).

The composed name has the form:

thread_name_prefix##[index]AWs where [index] is an integer that identifies the thread relative to the **DDS_AsyncWaitSet** (p. 1070).

If NULL, the default prefix will be used.

[default] NULL (use default prefix)

6.7.2.5 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 **DDS_AsyncWaitSet** (p. 1070) waits for an attached **DDS_Condition** (p. 1370) to trigger before it wakes up.

Duration must be a value greater than zero.

See also

DDS_WaitSet_wait (p. 1382)

[default] **DDS_DURATION_INFINITE** (p. 1208)

6.7.2.6 level

```
DDS_Long DDS_AsyncWaitSetProperty_t::level
```

Specifies the level of an **DDS_AsyncWaitSet** (p. 1070).

The level prevents an application to deadlock when it uses multiple **DDS_AsyncWaitSet** (p. 1070) 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 **DDS_AsyncWaitSet** (p. 1070) of bigger level.

[default] 1

6.8 DDS_AvailabilityQosPolicy Struct Reference

<<**extension**>> (p. 795) Configures the availability of data.

Data Fields

- **DDS_Boolean enable_required_subscriptions**
 <<**extension**>> (p. 795) Enables support for required subscriptions in a **DDS_DataWriter** (p. 485)
- struct **DDS_Duration_t max_data_availability_waiting_time**
 <<**extension**>> (p. 795) 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. 795) 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. 795) A sequence of endpoint groups.

6.8.1 Detailed Description

<<**extension**>> (p. 795) Configures the availability of data.

Entity:

DDS_DataReader (p. 615), **DDS_DataWriter** (p. 485)

Properties:

RxO (p. 1245) = NO

Changeable (p. 1245) = **YES** (p. 1245) (only on a **DDS_DataWriter** (p. 485) except for the member **DDS_↔AvailabilityQosPolicy::enable_required_subscriptions** (p. 1550))

6.8.2 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. 1759) set to **DDS_↔PERSISTENT_DURABILITY_QOS** (p. 1288) or **DDS_TRANSIENT_DURABILITY_QOS** (p. 1288) 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. 1890) `access_scope` set to **DDS_GROUP_PRESENTATION_QOS** (p. 1305). 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. 1305).

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. 1893) is set to **DDS_TOPIC_PRESENTATION_QOS** (p. 1305) or **DDS_INSTANCE_PRESENTATION_QOS** (p. 1305), the virtual HB contains information about the samples contained in the **DDS_DataWriter** (p. 485) history.

When **DDS_PresentationQosPolicy::access_scope** (p. 1893) is set to **DDS_GROUP_PRESENTATION_QOS** (p. 1305), the virtual HB contains information about all DataWriters in the **DDS_Publisher** (p. 443).

The frequency at which virtual HBs are sent is controlled by the protocol parameters **DDS_RtpsReliableWriterProtocol_t::virtual_heartbeat_period** (p. 1965) and **DDS_RtpsReliableWriterProtocol_t::samples_per_virtual_heartbeat** (p. 1965).

Required Subscriptions

In the context of Required Subscriptions, this QoS policy can be used to configure a set of Required Subscriptions on a **DDS_DataWriter** (p. 485).

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.

6.8.3 Consistency

For a DataWriter, the setting of **AVAILABILITY** (p. 1250) must be set consistently with that of the **RELIABILITY** (p. 1321) and **DURABILITY** (p. 1285).

If **DDS_AvailabilityQosPolicy::enable_required_subscriptions** (p. 1550) is set to **DDS_BOOLEAN_TRUE** (p. 1201), **DDS_ReliabilityQosPolicy::kind** (p. 1939) must be set to **DDS_RELIABLE_RELIABILITY_QOS** (p. 1323), **DDS_DurabilityQosPolicy** (p. 1759) must be set to a value different than **DDS_VOLATILE_DURABILITY_QOS** (p. 1287), and **DDS_DurabilityQosPolicy::writer_depth** (p. 1762) must be set to either **DDS_AUTO_WRITER_DEPTH** (p. 1288) or **DDS_LENGTH_UNLIMITED** (p. 1326).

6.8.4 Field Documentation

6.8.4.1 enable_required_subscriptions

DDS_Boolean DDS_AvailabilityQosPolicy::enable_required_subscriptions

<<**extension**>> (p. 795) Enables support for required subscriptions in a **DDS_DataWriter** (p. 485)

[default] **DDS_BOOLEAN_FALSE** (p. 1201)

Categories

Immutable (p. 914), **Reliability** (p. 895), **Durability** (p. 860), **Data Availability** (p. 843)

6.8.4.2 max_data_availability_waiting_time

struct DDS_Duration_t DDS_AvailabilityQosPolicy::max_data_availability_waiting_time

<<**extension**>> (p. 795) 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. 1551) have been discovered or **max_endpoint_availability_waiting_time** (p. 1550) has expired.

Required Subscriptions

This field is not applicable to Required Subscriptions.

[default] **DDS_DURATION_AUTO** (p. 1209) (**DDS_DURATION_INFINITE** (p. 1208) for **DDS_GROUP_PRESENTATION↔_QOS** (p. 1305). Otherwise, 0 seconds)

[range] [0, **DDS_DURATION_INFINITE** (p. 1208)], **DDS_DURATION_AUTO** (p. 1209)

Categories

Mutable (p. 949), **Reliability** (p. 895), **Durability** (p. 860), **Data Availability** (p. 843)

6.8.4.3 max_endpoint_availability_waiting_time

```
struct DDS_Duration_t DDS_AvailabilityQosPolicy::max_endpoint_availability_waiting_time
```

<<**extension**>> (p. 795) 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. 1551).

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. 1551) are discovered or this timeout expires.

Required Subscriptions

This field is not applicable to Required Subscriptions.

[default] **DDS_DURATION_AUTO** (p. 1209) (**DDS_DURATION_INFINITE** (p. 1208) for **DDS_GROUP_PRESENTATION_QOS** (p. 1305). Otherwise, 0 seconds)

[range] [0, **DDS_DURATION_INFINITE** (p. 1208)], **DDS_DURATION_AUTO** (p. 1209)

Categories

Mutable (p. 949), **Reliability** (p. 895), **Durability** (p. 860), **Data Availability** (p. 843)

6.8.4.4 required_matched_endpoint_groups

```
struct DDS_EndpointGroupSeq DDS_AvailabilityQosPolicy::required_matched_endpoint_groups
```

<<**extension**>> (p. 795) 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. 1672).

Required Subscriptions

In the context of Required Subscriptions, it specifies the set of Required Subscriptions on a **DDS_DataWriter** (p. 485).

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. 1617).

[default] Empty sequence

Categories

Mutable (p. 949), **Reliability** (p. 895), **Durability** (p. 860), **Data Availability** (p. 843)

6.9 DDS_BatchQosPolicy Struct Reference

<<**extension**>> (p. 795) Used to configure batching of multiple samples into a single network packet in order to increase throughput for small samples.

Data Fields

- **DDS_Boolean enable**
 <<**basic**>> (p. 783) <<**extension**>> (p. 795) *Specifies whether or not batching is enabled.*
- **DDS_Long max_data_bytes**
 <<**basic**>> (p. 783) <<**extension**>> (p. 795) *The maximum cumulative length of all serialized samples in a batch.*
- **DDS_Long max_samples**
 <<**basic**>> (p. 783) <<**extension**>> (p. 795) *The maximum number of samples in a batch.*
- struct **DDS_Duration_t max_flush_delay**
 <<**extension**>> (p. 795) *The maximum flush delay.*
- struct **DDS_Duration_t source_timestamp_resolution**
 <<**extension**>> (p. 795) *Batch source timestamp resolution.*
- **DDS_Boolean thread_safe_write**
 <<**extension**>> (p. 795) *Determines whether or not the write operation is thread safe.*

6.9.1 Detailed Description

<<**extension**>> (p. 795) 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:

DDS_DataWriter (p. 485)

Properties:

RxO (p. 1245) = NO

Changeable (p. 1245) = **UNTIL ENABLE** (p. 1245)

6.9.2 Field Documentation

6.9.2.1 enable

`DDS_Boolean DDS_BatchQosPolicy::enable`

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Specifies whether or not batching is enabled.

[default] `DDS_BOOLEAN_FALSE` (p. 1201)

Categories

Immutable (p. 914), **Batching** (p. 838), **Throughput** (p. 908)

6.9.2.2 max_data_bytes

`DDS_Long DDS_BatchQosPolicy::max_data_bytes`

<<**basic**>> (p. 783) <<**extension**>> (p. 795) 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. 1326)]

6.9.3 Consistency

The setting of `DDS_BatchQosPolicy::max_data_bytes` (p. 1553) must be consistent with `DDS_BatchQosPolicy::max_samples` (p. 1553). For these two values to be consistent, they cannot be both `DDS_LENGTH_UNLIMITED` (p. 1326).

Categories

Immutable (p. 914), **Batching** (p. 838), **Throughput** (p. 908)

6.9.3.1 max_samples

`DDS_Long DDS_BatchQosPolicy::max_samples`

<<**basic**>> (p. 783) <<**extension**>> (p. 795) The maximum number of samples in a batch.

A batch is flushed automatically when this maximum is reached.

[default] `DDS_LENGTH_UNLIMITED` (p. 1326)

[range] [1, `DDS_LENGTH_UNLIMITED` (p. 1326)]

6.9.4 Consistency

The setting of `DDS_BatchQosPolicy::max_samples` (p. 1553) must be consistent with `DDS_BatchQosPolicy::max_data_bytes` (p. 1553). For these two values to be consistent, they cannot be both `DDS_LENGTH_UNLIMITED` (p. 1326).

Categories

Immutable (p. 914), **Batching** (p. 838), **Throughput** (p. 908)

6.9.4.1 max_flush_delay

`struct DDS_Duration_t DDS_BatchQosPolicy::max_flush_delay`

<<**extension**>> (p. 795) 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. 1208)

[range] [0, `DDS_DURATION_INFINITE` (p. 1208)]

6.9.5 Consistency

The setting of `DDS_BatchQosPolicy::max_flush_delay` (p. 1554) must be consistent with `DDS_AsynchronousPublisherQosPolicy::disable_asynchronous_batch` (p. 1541) and `DDS_BatchQosPolicy::thread_safe_write` (p. 1555). If the delay is different than `DDS_DURATION_INFINITE` (p. 1208), `DDS_AsynchronousPublisherQosPolicy::disable_asynchronous_batch` (p. 1541) must be set to `DDS_BOOLEAN_FALSE` (p. 1201) and `DDS_BatchQosPolicy::thread_safe_write` (p. 1555) must be set to `DDS_BOOLEAN_TRUE` (p. 1201).

Categories

Immutable (p. 914), **Batching** (p. 838), **Throughput** (p. 908)

6.9.5.1 source_timestamp_resolution

```
struct DDS_Duration_t DDS_BatchQosPolicy::source_timestamp_resolution
```

<<**extension**>> (p. 795) 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. 1208), 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. 1208).

[default] **DDS_DURATION_INFINITE** (p. 1208)

[range] [0, **DDS_DURATION_INFINITE** (p. 1208)]

6.9.6 Consistency

The setting of **DDS_BatchQosPolicy::source_timestamp_resolution** (p. 1554) must be consistent with **DDS_BatchQosPolicy::thread_safe_write** (p. 1555). If **DDS_BatchQosPolicy::thread_safe_write** (p. 1555) is set to **DDS_BOOLEAN_FALSE** (p. 1201), **DDS_BatchQosPolicy::source_timestamp_resolution** (p. 1554) must be set to **DDS_DURATION_INFINITE** (p. 1208).

Categories

Immutable (p. 914), **Batching** (p. 838), **Throughput** (p. 908), **Data Availability** (p. 843)

6.9.6.1 thread_safe_write

```
DDS_Boolean DDS_BatchQosPolicy::thread_safe_write
```

<<**extension**>> (p. 795) Determines whether or not the write operation is thread safe.

If this parameter is set to **DDS_BOOLEAN_TRUE** (p. 1201), multiple threads can call write on the **DDS_DataWriter** (p. 485) concurrently.

[default] **DDS_BOOLEAN_TRUE** (p. 1201)

6.9.7 Consistency

The setting of `DDS_BatchQosPolicy::thread_safe_write` (p. 1555) must be consistent with `DDS_BatchQosPolicy::source_timestamp_resolution` (p. 1554). If `DDS_BatchQosPolicy::thread_safe_write` (p. 1555) is set to `DDS_BOOLEAN_FALSE` (p. 1201), `DDS_BatchQosPolicy::source_timestamp_resolution` (p. 1554) must be set to `DDS_DURATION_INFINITE` (p. 1208).

Categories

Immutable (p. 914), **Batching** (p. 838), **Throughput** (p. 908), **Concurrency** (p. 839)

6.10 DDS_BooleanSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_Boolean** (p. 1204) >

6.10.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_Boolean** (p. 1204) >

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_Boolean (p. 1204)

FooSeq (p. 2113)

6.11 DDS_BuiltinTopicKey_t Struct Reference

The key type of the built-in topic types.

Data Fields

- `DDS_BUILTIN_TOPIC_KEY_TYPE_NATIVE` **value** [`DDS_BUILTIN_TOPIC_KEY_TYPE_NATIVE_LENGTH`]
An array of four integers that uniquely represents a remote **DDS_Entity** (p. 1360).

6.11.1 Detailed Description

The key type of the built-in topic types.

Each remote **DDS_Entity** (p. 1360) 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. 1870)

DDS_TopicBuiltinTopicData (p. 2034)

DDS_PublicationBuiltinTopicData (p. 1905)

DDS_SubscriptionBuiltinTopicData (p. 2012)

6.12 DDS_BuiltinTopicReaderResourceLimits_t Struct Reference

<<*extension*>> (p. 795) Built-in topic reader's resource limits.

Data Fields

- **DDS_Long initial_samples**
<<*extension*>> (p. 795) Initial number of samples.
- **DDS_Long max_samples**
<<*extension*>> (p. 795) Maximum number of samples.
- **DDS_Long initial_infos**
<<*extension*>> (p. 795) Initial number of sample infos.
- **DDS_Long max_infos**
<<*extension*>> (p. 795) Maximum number of sample infos.
- **DDS_Long initial_outstanding_reads**
<<*extension*>> (p. 795) The initial number of outstanding reads that have not called finish yet on the same built-in topic **DDS_DataReader** (p. 615).
- **DDS_Long max_outstanding_reads**
<<*extension*>> (p. 795) The maximum number of outstanding reads that have not called finish yet on the same built-in topic **DDS_DataReader** (p. 615).
- **DDS_Long max_samples_per_read**
<<*extension*>> (p. 795) Maximum number of samples that can be read/taken on a same built-in topic **DDS_DataReader** (p. 615).
- **DDS_Boolean disable_fragmentation_support**
[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Determines whether the built-in topic **DDS_DataReader** (p. 615) can receive fragmented samples.
- **DDS_Long max_fragmented_samples**
<<*extension*>> (p. 795) The maximum number of samples for which the built-in topic **DDS_DataReader** (p. 615) may store fragments at a given point in time.
- **DDS_Long initial_fragmented_samples**
<<*extension*>> (p. 795) The initial number of samples for which a built-in topic **DDS_DataReader** (p. 615) may store fragments.

- **DDS_Long max_fragmented_samples_per_remote_writer**
 <<extension>> (p. 795) The maximum number of samples per remote writer for which a built-in topic **DDS_DataReader** (p. 615) may store fragments.
- **DDS_Long max_fragments_per_sample**
 <<extension>> (p. 795) Maximum number of fragments for a single sample.
- **DDS_Boolean dynamically_allocate_fragmented_samples**
 <<extension>> (p. 795) Determines whether the built-in topic **DDS_DataReader** (p. 615) pre-allocates storage for storing fragmented samples.

6.12.1 Detailed Description

<<extension>> (p. 795) 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. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

QoS:

DDS_DiscoveryConfigQosPolicy (p. 1689)

6.12.2 Field Documentation

6.12.2.1 initial_samples

DDS_Long DDS_BuiltinTopicReaderResourceLimits_t::initial_samples

<<extension>> (p. 795) 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. 914), **Sample Memory Management** (p. 902), **Middleware Memory Management** (p. 878), **Durability** (p. 860)

6.12.2.2 max_samples

DDS_Long DDS_BuiltinTopicReaderResourceLimits_t::max_samples

<<**extension**>> (p. 795) 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. 1326)

[range] [1, 1 million] or **DDS_LENGTH_UNLIMITED** (p. 1326), >= initial_samples

Categories

Immutable (p. 914), **Sample Memory Management** (p. 902), **Middleware Memory Management** (p. 878), **Durability** (p. 860)

6.12.2.3 initial_infos

DDS_Long DDS_BuiltinTopicReaderResourceLimits_t::initial_infos

<<**extension**>> (p. 795) Initial number of sample infos.

The initial number of info units that a built-in topic **DDS_DataReader** (p. 615) can have. Info units are used to store **DDS_SampleInfo** (p. 1985).

[default] 64

[range] [1, 1 million] <= max_infos

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.12.2.4 max_infos

DDS_Long DDS_BuiltinTopicReaderResourceLimits_t::max_infos

<<**extension**>> (p. 795) Maximum number of sample infos.

The maximum number of info units that a built-in topic **DDS_DataReader** (p. 615) can use to store **DDS_SampleInfo** (p. 1985).

[default] DDS_LENGTH_UNLIMITED (p. 1326)

[range] [1, 1 million] or **DDS_LENGTH_UNLIMITED** (p. 1326), >= initial_infos

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.12.2.5 initial_outstanding_reads

DDS_Long DDS_BuiltinTopicReaderResourceLimits_t::initial_outstanding_reads

<<**extension**>> (p. 795) The initial number of outstanding reads that have not called finish yet on the same built-in topic **DDS_DataReader** (p. 615).

Must be less than or equal to max_outstanding_reads.

[default] 2

[range] [1, 1024]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Data Availability** (p. 843)

6.12.2.6 max_outstanding_reads

DDS_Long DDS_BuiltinTopicReaderResourceLimits_t::max_outstanding_reads

<<**extension**>> (p. 795) The maximum number of outstanding reads that have not called finish yet on the same built-in topic **DDS_DataReader** (p. 615).

Must be greater than or equal to initial_outstanding_reads.

[default] DDS_LENGTH_UNLIMITED (p. 1326)

[range] [1, 1024] or **DDS_LENGTH_UNLIMITED** (p. 1326)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Data Availability** (p. 843)

6.12.2.7 max_samples_per_read

DDS_Long DDS_BuiltinTopicReaderResourceLimits_t::max_samples_per_read

<<**extension**>> (p. 795) Maximum number of samples that can be read/taken on a same built-in topic **DDS_DataReader** (p. 615).

[default] 1024

[range] [1, 65536]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Data Availability** (p. 843)

6.12.2.8 disable_fragmentation_support

DDS_Boolean DDS_BuiltinTopicReaderResourceLimits_t::disable_fragmentation_support

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Determines whether the built-in topic **DDS_DataReader** (p. 615) can receive fragmented samples.

When fragmentation support is not needed, disabling fragmentation support will save some memory resources.

[default] **DDS_BOOLEAN_FALSE** (p. 1201)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Fragmentation** (p. 866), **Sending Large Data** (p. 871)

6.12.2.9 max_fragmented_samples

DDS_Long DDS_BuiltinTopicReaderResourceLimits_t::max_fragmented_samples

<<*extension*>> (p. 795) The maximum number of samples for which the built-in topic **DDS_DataReader** (p. 615) may store fragments at a given point in time.

At any given time, a built-in topic **DDS_DataReader** (p. 615) 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 **DDS_DataWriter** (p. 485) 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. 1558).

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. 1560) is **DDS_BOOLEAN_FALSE** (p. 1201).

[default] 1024

[range] [1, 1 million]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Fragmentation** (p. 866), **Sending Large Data** (p. 871)

6.12.2.10 initial_fragmented_samples

DDS_Long DDS_BuiltinTopicReaderResourceLimits_t::initial_fragmented_samples

<<**extension**>> (p. 795) The initial number of samples for which a built-in topic **DDS_DataReader** (p. 615) may store fragments.

Only applies if **DDS_BuiltinTopicReaderResourceLimits_t::disable_fragmentation_support** (p. 1560) is **DDS_↵
BOOLEAN_FALSE** (p. 1201).

[default] 4

[range] [1,1024], <= max_fragmented_samples

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Fragmentation** (p. 866), **Sending Large Data** (p. 871)

6.12.2.11 max_fragmented_samples_per_remote_writer

DDS_Long DDS_BuiltinTopicReaderResourceLimits_t::max_fragmented_samples_per_remote_writer

<<**extension**>> (p. 795) The maximum number of samples per remote writer for which a built-in topic **DDS_Data↵
Reader** (p. 615) 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. 1560) is **DDS_↵
BOOLEAN_FALSE** (p. 1201).

[default] 256

[range] [1, 1 million], <= max_fragmented_samples

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Fragmentation** (p. 866), **Sending Large Data** (p. 871)

6.12.2.12 max_fragments_per_sample

DDS_Long DDS_BuiltinTopicReaderResourceLimits_t::max_fragments_per_sample

<<**extension**>> (p. 795) Maximum number of fragments for a single sample.

Only applies if **DDS_BuiltinTopicReaderResourceLimits_t::disable_fragmentation_support** (p. 1560) is **DDS_↵
BOOLEAN_FALSE** (p. 1201).

[default] **DDS_LENGTH_UNLIMITED** (p. 1326)

[range] [1, 1 million] or **DDS_LENGTH_UNLIMITED** (p. 1326)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Fragmentation** (p. 866), **Sending Large Data** (p. 871)

6.12.2.13 dynamically_allocate_fragmented_samples

DDS_Boolean DDS_BuiltinTopicReaderResourceLimits_t::dynamically_allocate_fragmented_samples

<<**extension**>> (p. 795) Determines whether the built-in topic **DDS_DataReader** (p. 615) 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. 1201), the middleware will allocate memory upfront for storing fragments for up to **DDS_DataReaderResourceLimitsQosPolicy↵
::initial_fragmented_samples** (p. 1627) samples. This memory may grow up to **DDS_DataReaderResourceLimits↵
QosPolicy::max_fragmented_samples** (p. 1626) if needed.

Only applies if **DDS_DataReaderResourceLimitsQosPolicy::disable_fragmentation_support** (p. 1626) is **DDS_↵
BOOLEAN_FALSE** (p. 1201).

[default] **DDS_BOOLEAN_TRUE** (p. 1201)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Fragmentation** (p. 866), **Sending Large Data** (p. 871)

6.13 DDS_ChannelSettings_t Struct Reference

<<*extension*>> (p. 795) Type used to configure the properties of a channel.

Data Fields

- struct **DDS_TransportMulticastSettingsSeq** **multicast_settings**
 <<*extension*>> (p. 795) A sequence of **DDS_TransportMulticastSettings_t** (p. 2060) used to configure the multicast addresses associated with a channel.
- char * **filter_expression**
 <<*extension*>> (p. 795) A logical expression used to determine the data that will be published in the channel.
- **DDS_Long** **priority**
 <<*extension*>> (p. 795) Publication priority

6.13.1 Detailed Description

<<*extension*>> (p. 795) Type used to configure the properties of a channel.

QoS:

DDS_MultiChannelQosPolicy (p. 1856)

6.13.2 Field Documentation

6.13.2.1 multicast_settings

```
struct DDS_TransportMulticastSettingsSeq DDS_ChannelSettings_t::multicast_settings
```

<<*extension*>> (p. 795) A sequence of **DDS_TransportMulticastSettings_t** (p. 2060) 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. 1901) associated with the **DDS_DomainParticipantQos** (p. 1726).

[default] Empty sequence (invalid value)

Categories

Immutable (p. 914), **Transports** (p. 908), **Content Filtering** (p. 841)

6.13.2.2 filter_expression

char* DDS_ChannelSettings_t::filter_expression

<<**extension**>> (p. 795) 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. 1858)

Important: This value must be an allocated string with **DDS_String_alloc** (p. 1528) or **DDS_String_dup** (p. 1528). It should not be assigned to a string constant.

The filter expression length (including NULL-terminated character) cannot be greater than **DDS_DomainParticipantResourceLimitsQosPolicy::channel_filter_expression_max_length** (p. 1752).

See also

Queries and Filters Syntax (p. 738)

[default] NULL (invalid value)

Categories

Immutable (p. 914), **Content Filtering** (p. 841)

6.13.2.3 priority

DDS_Long DDS_ChannelSettings_t::priority

<<**extension**>> (p. 795) 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. 1319)) with **DDS_FlowControllerProperty_t::scheduling_policy** (p. 1799) set to **DDS_HPF_FLOW**↔**CONTROLLER_SCHED_POLICY** (p. 561).

Larger numbers have higher priority.

If the publication priority of the channel is any value other than **DDS_PUBLICATION_PRIORITY_UNDEFINED** (p. 1318), 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. 1318), then the channel's priority will be set to the value of the data writer's priority.

If the publication priority of both the data writer and the channel are **DDS_PUBLICATION_PRIORITY_UNDEFINED** (p. 1318), the channel will be assigned the lowest priority value.

If the publication priority of the channel is **DDS_PUBLICATION_PRIORITY_AUTOMATIC** (p. 1318), 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. 2101) of the **FooDataWriter_write_w_params** (p. 501) function.

[default] **DDS_PUBLICATION_PRIORITY_UNDEFINED** (p. 1318)

Categories

Immutable (p. 914), **Transports** (p. 908), **Data Availability** (p. 843)

6.14 DDS_ChannelSettingsSeq Struct Reference

Declares IDL sequence< **DDS_ChannelSettings_t** (p. 1564) >

6.14.1 Detailed Description

Declares IDL sequence< **DDS_ChannelSettings_t** (p. 1564) >

A sequence of **DDS_ChannelSettings_t** (p. 1564) used to configure the channels' properties. If the length of the sequence is zero, the **DDS_MultiChannelQosPolicy** (p. 1856) has no effect.

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_ChannelSettings_t (p. 1564)

6.15 DDS_CharSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_Char** (p. 1201) >

6.15.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_Char** (p. 1201) >

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_Char (p. 1201)

FooSeq (p. 2113)

6.16 DDS_CoherentSetInfo_t Struct Reference

<<*extension*>> (p. 1012) Type definition for a coherent set info.

Data Fields

- **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 **DDS_DataWriter** (p. 485) 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.

6.16.1 Detailed Description

<<**extension**>> (p. 1012) Type definition for a coherent set info.

A CoherentSetInfo provides information about the coherent set associated with a sample.

6.16.2 Field Documentation

6.16.2.1 group_guid

```
DDS_GUID_t DDS_CoherentSetInfo_t::group_guid
```

The coherent set group GUID.

This GUID identifies the **DDS_DataWriter** (p. 485) or the group of DataWriters publishing the coherent set, depending on the value of **DDS_PresentationQosPolicy::access_scope** (p. 1893) in the **DDS_Subscriber** (p. 572).

6.16.2.2 coherent_set_sequence_number

```
DDS_SequenceNumber_t DDS_CoherentSetInfo_t::coherent_set_sequence_number
```

Sequence number that identifies a sample as part of a **DDS_DataWriter** (p. 485) coherent set.

When **DDS_PresentationQosPolicy::access_scope** (p. 1893) in the **DDS_Subscriber** (p. 572) is set to **DDS_↔INSTANCE_PRESENTATION_QOS** (p. 1305) or **DDS_TOPIC_PRESENTATION_QOS** (p. 1305), the coherent set associated with a sample is identified by the pair (group_guid, coherent_set_sequence_number).

6.16.2.3 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. 1893) in the `DDS_Subscriber` (p. 572) is set to `DDS_GROUP_PRESENTATION_QOS` (p. 1305), the coherent set associated with a sample is identified by the pair (group_↵_guid, group_coherent_set_sequence_number).

6.16.2.4 incomplete_coherent_set

`DDS_Boolean DDS_CoherentSetInfo_t::incomplete_coherent_set`

Indicates if a sample is part of an incomplete coherent set.

6.17 DDS_CompressionSettings_t Struct Reference

<<*extension*>> (p. 795) Settings related to compressing user data.

Data Fields

- **DDS_CompressionIdMask compression_ids**
 <<*extension*>> (p. 795) Mask that represents the compression algorithms enabled.
- **DDS_UnsignedLong writer_compression_level**
 <<*extension*>> (p. 795) The level of compression to use when compressing data.
- **DDS_Long writer_compression_threshold**
 <<*extension*>> (p. 795) The threshold, in bytes, above which a serialized sample will be eligible to be compressed.

6.17.1 Detailed Description

<<*extension*>> (p. 795) Settings related to compressing user data.

QoS:

`DDS_DataRepresentationQosPolicy` (p. 1638)

6.17.2 Field Documentation

6.17.2.1 compression_ids

DDS_CompressionIdMask DDS_CompressionSettings_t::compression_ids

<<**extension**>> (p. 795) Mask that represents the compression algorithms enabled.

A bitmap that represents the compression algorithm IDs (**DDS_CompressionIdMask** (p. 1262)) that are supported by the endpoint. The **DDS_DataWriter** (p. 485) creation will fail if more than one algorithm is provided.

If a **DDS_DataWriter** (p. 485) inherits multiple compression IDs from a **DDS_Topic** (p. 175), only the least significant bit enabled will be inherited. This forces the following order of preference: **DDS_COMPRESSION_ID_ZLIB** (p. 1262), **DDS_COMPRESSION_ID_BZIP2** (p. 1262), **DDS_COMPRESSION_ID_LZ4** (p. 1263).

Interactions with Security and Batching: Currently, the only algorithm that is supported when compression and batching are enabled on the same **DDS_DataWriter** (p. 485) is **DDS_COMPRESSION_ID_ZLIB** (p. 1262).

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. 1652) is enabled and a dispose message is sent, the serialized key is not compressed.

[default] For **DDS_Topic** (p. 175), **DDS_DataWriter** (p. 485) a **DDS_CompressionIdMask** (p. 1262) mask set to **DDS_COMPRESSION_ID_MASK_NONE** (p. 1260)

[default] For **DDS_DataReader** (p. 615) a **DDS_CompressionIdMask** (p. 1262) mask set to **DDS_COMPRESSION_ID_MASK_ALL** (p. 1260).

Categories

Immutable (p. 914), **Matching** (p. 963), **Discovery Metadata** (p. 958), **Sending Large Data** (p. 871)

6.17.2.2 writer_compression_level

DDS_UnsignedLong DDS_CompressionSettings_t::writer_compression_level

<<**extension**>> (p. 795) 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. 1260)

Note

Only available for a **DDS_DataWriter** (p. 485) and **DDS_Topic** (p. 175).

Categories

Immutable (p. 914), **Sending Large Data** (p. 871)

6.17.2.3 writer_compression_threshold

DDS_Long DDS_CompressionSettings_t::writer_compression_threshold

<<**extension**>> (p. 795) 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. 1553)

[range] [0, 2147483647] or **DDS_LENGTH_UNLIMITED** (p. 1326)

Setting the threshold to **DDS_LENGTH_UNLIMITED** (p. 1326) disables the compression.

[default] **DDS_COMPRESSION_THRESHOLD_DEFAULT** (p. 1261) (8192)

Note

Only available for a **DDS_DataWriter** (p. 485) and **DDS_Topic** (p. 175).

Categories

Immutable (p. 914), **Sending Large Data** (p. 871)

6.18 DDS_ConditionHandler Struct Reference

<<**extension**>> (p. 1012) <<**interface**>> (p. 1013) Handler called by the **DDS_Condition_dispatch** (p. 1375).

Data Fields

- void * **handler_data**
A place for handler implementors to keep a pointer to data that may be needed by their handler.
- **DDS_ConditionHandler_OnConditionTriggeredCallback** **on_condition_triggered**
*Handles the dispatch of a **DDS_Condition** (p. 1370).*

6.18.1 Detailed Description

<<**extension**>> (p. 1012) <<**interface**>> (p. 1013) Handler called by the **DDS_Condition_dispatch** (p. 1375).

Direct known implementations: **DDS_DataReaderStatusConditionHandler** (p. 1075)

6.18.2 Field Documentation

6.18.2.1 handler_data

```
void* DDS_ConditionHandler::handler_data
```

A place for handler implementors to keep a pointer to data that may be needed by their handler.

6.18.2.2 on_condition_triggered

```
DDS_ConditionHandler_OnConditionTriggeredCallback DDS_ConditionHandler::on_condition_triggered
```

Handles the dispatch of a **DDS_Condition** (p. 1370).

This callback is called by **DDS_Condition_dispatch** (p. 1375).

6.19 DDS_ConditionSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_Condition** (p. 1370) >

6.19.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_Condition** (p. 1370) >

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_WaitSet (p. 1371)

FooSeq (p. 2113)

6.20 DDS_ContentFilter Struct Reference

<<*interface*>> (p. 1013) Interface to be used by a custom filter of a **DDS_ContentFilteredTopic** (p. 177)

Data Fields

- **DDS_ContentFilterCompileFunction compile**
Compile an instance of the content filter according to the filter expression and parameters of the given data type.
- **DDS_ContentFilterWriterCompileFunction writer_compile**
*A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching **DDS_DataReader** (p. 615).*
- **DDS_ContentFilterEvaluateFunction evaluate**
Evaluate whether the sample is passing the filter or not according to the sample content.
- **DDS_ContentFilterWriterEvaluateFunction writer_evaluate**
A writer-side filtering API to retrieve a list of DataReaders whose content filters pass the sample.
- **DDS_ContentFilterFinalizeFunction finalize**
A previously compiled instance of the content filter is no longer in use and resources can now be cleaned up.
- **DDS_ContentFilterWriterFinalizeFunction writer_finalize**
A writer-side filtering API to clean up a previously compiled instance of the content filter.
- **DDS_ContentFilterWriterAttachFunction writer_attach**
A writer-side filtering API to create some state that can facilitate filtering on the writer side.
- **DDS_ContentFilterWriterDetachFunction writer_detach**
*A writer-side filtering API to clean up a previously created state using **DDS_ContentFilter::writer_attach** (p. 1577).*
- **DDS_ContentFilterWriterReturnLoanFunction writer_return_loan**
*A writer-side filtering API to return the loan on the list of DataReaders returned by **DDS_ContentFilter::writer_evaluate** (p. 1576).*
- void * **filter_data**
A place for filter implementors to keep a pointer to data that may be needed by their filter.

6.20.1 Detailed Description

<<**interface**>> (p. 1013) Interface to be used by a custom filter of a **DDS_ContentFilteredTopic** (p. 177)

Entity:

DDS_ContentFilteredTopic (p. 177)

This interface can be implemented by an application-provided class and then registered with the **DDS_DomainParticipant** (p. 77) such that samples can be filtered for a **DDS_ContentFilteredTopic** (p. 177) with the given filter name.

Note: the API for using a custom content filter is subject to change in a future release.

See also

DDS_ContentFilteredTopic (p. 177)

DDS_DomainParticipant_register_contentfilter (p. 120)

6.20.2 Field Documentation

6.20.2.1 compile

DDS_ContentFilterCompileFunction DDS_ContentFilter::compile

Compile an instance of the content filter according to the filter expression and parameters of the given data type.

This function 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 **DDS_ContentFilteredTopic** (p. 177) with the matching filter name is created, or when a **DDS_DataReader** (p. 615) 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

<i>new_compile_data</i>	<< out >> (p. 1013) User specified opaque pointer of this instance of the content filter. This value is then passed to the DDS_ContentFilter::evaluate (p. 1574) and DDS_ContentFilter::finalize (p. 1576) functions for this instance of the content filter. Can be set to NULL .
<i>expression</i>	<< in >> (p. 1013) An ASCIIZ string with the filter expression. The memory used by this string is owned by RTI Connex and must not be freed. If you want to manipulate this string, you must first make a copy of it.
<i>parameters</i>	<< in >> (p. 1013) A string sequence with the expression parameters the DDS_ContentFilteredTopic (p. 177) was created with. The string sequence is equal (but not identical) to the string sequence passed to DDS_DomainParticipant_create_contentfilteredtopic (p. 118). Note that the sequence passed to the compile function is owned by RTI Connex and must not be referenced outside the compile function.
<i>type_code</i>	<< in >> (p. 1013) A pointer to the type code for the related DDS_Topic (p. 175) of the DDS_ContentFilteredTopic (p. 177). A type_code is a description of a type in terms of which types it contains (such as long, 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.
<i>type_class_name</i>	<< in >> (p. 1013) Fully qualified class name of the related DDS_Topic (p. 175).
<i>old_compile_data</i>	<< in >> (p. 1013) 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 DDS_ContentFilteredTopic (p. 177), 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.

Returns

One of the **Standard Return Codes** (p. 1221)

6.20.2.2 writer_compile

DDS_ContentFilterWriterCompileFunction DDS_ContentFilter::writer_compile

A writer-side filtering API to compile an instance of the content filter according to the filter expression and parameters specified by a matching **DDS_DataReader** (p. 615).

This function is called when the **DDS_DataWriter** (p. 485) discovers a **DDS_DataReader** (p. 615) with a **DDS_ContentFilteredTopic** (p. 177) or when a **DDS_DataWriter** (p. 485) 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

<i>writer_filter_data</i>	<< <i>in</i> >> (p. 1013) A pointer to the state created using DDS_ContentFilter::writer_attach (p. 1577) .
<i>prop</i>	<< <i>out</i> >> (p. 1013) A pointer to DDS_ExpressionProperty (p. 1795) that allows you to indicate to RTI Connex if a filter expression can be optimized.
<i>expression</i>	<< <i>in</i> >> (p. 1013) An ASCII string with the filter expression. The memory used by this string is owned by RTI Connex and must not be freed. If you want to manipulate this string, you must first make a copy of it.
<i>parameters</i>	<< <i>in</i> >> (p. 1013) A string sequence with the expression parameters with which the DDS_ContentFilteredTopic (p. 177) was created. The string sequence is equal (but not identical) to the string sequence passed to DDS_DomainParticipant_create_contentfilteredtopic (p. 118). Note that the sequence passed to the compile function is owned by RTI Connex and must not be referenced outside the compile function.
<i>type_code</i>	<< <i>in</i> >> (p. 1013) A pointer to the type code for the related DDS_Topic (p. 175) of the DDS_ContentFilteredTopic (p. 177). A type_code is a description of a type in terms of which types it contains (such as long, 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.
<i>type_class_name</i>	<< <i>in</i> >> (p. 1013) Fully qualified class name of the related DDS_Topic (p. 175).
<i>cookie</i>	<< <i>in</i> >> (p. 1013) DDS_Cookie_t (p. 1580) to uniquely identify DDS_DataReader (p. 615) for which DDS_ContentFilter::writer_compile (p. 1573) was called.

Returns

One of the **Standard Return Codes** (p. 1221)

6.20.2.3 evaluate

DDS_ContentFilterEvaluateFunction DDS_ContentFilter::evaluate

Evaluate whether the sample is passing the filter or not according to the sample content.

This function is called when a sample for a locally created **DDS_DataReader** (p. 615) associated with the filter is received, or when a sample for a discovered **DDS_DataReader** (p. 615) associated 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

<i>compile_data</i>	<< <i>in</i> >> (p. 1013) The last return value of the DDS_ContentFilter::compile (p. 1572) function for this instance of the content filter. Can be NULL .
<i>sample</i>	<< <i>in</i> >> (p. 1013) Pointer to a deserialized sample to be filtered
<i>meta_data</i>	<< <i>in</i> >> (p. 1013) Pointer to meta data associated with the sample.

Returns

The function must return 0 if the sample should be filtered out, non zero otherwise

6.20.2.4 writer_evaluate

DDS_ContentFilterWriterEvaluateFunction DDS_ContentFilter::writer_evaluate

A writer-side filtering API to retrieve a list of DataReaders whose content filters pass the sample.

This function is called every time a **DDS_DataWriter** (p.485) writes a new sample. Its purpose is to evaluate the sample for all the readers for which the **DDS_DataWriter** (p.485) is performing writer-side filtering and return the list of **DDS_Cookie_t** (p. 1580) 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

<i>writer_filter_data</i>	<< <i>in</i> >> (p. 1013) A pointer to the state created using DDS_ContentFilter::writer_attach (p. 1577) .
<i>sample</i>	<< <i>in</i> >> (p. 1013) Pointer to a deserialized sample to be filtered.
<i>meta_data</i>	<< <i>in</i> >> (p. 1013) Pointer to meta data associated with the sample.

Returns

The function returns **DDS_CookieSeq** (p. 1581) which identifies the set of DataReaders whose filters pass the sample.

6.20.2.5 finalize

DDS_ContentFilterFinalizeFunction DDS_ContentFilter::finalize

A previously compiled instance of the content filter is no longer in use and resources can now be cleaned up.

This function 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 **DDS_ContentFilteredTopic** (p. 177) with the matching filter name is deleted, or when a **DDS_DataReader** (p. 615) with a matching filter name is removed due to discovery.

This function is also called on all instances of the discovered **DDS_DataReader** (p. 615) with a matching filter name if the filter is unregistered with **DDS_DomainParticipant_unregister_contentfilter** (p. 121).

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

<i>compile_data</i>	<< <i>in</i> >> (p. 1013) The last return value of the DDS_ContentFilter::compile (p. 1572) function for this instance of the content filter. Can be NULL

6.20.2.6 writer_finalize

DDS_ContentFilterWriterFinalizeFunction `DDS_ContentFilter::writer_finalize`

A writer-side filtering API to clean up a previously compiled instance of the content filter.

This function is called to notify the filter implementation that the **DDS_DataWriter** (p. 485) is no longer matching with a **DDS_DataReader** (p. 615) for which it was previously performing writer-side filtering. This will allow the filter to purge any state it was maintaining for the **DDS_DataReader** (p. 615).

It is possible for multiple threads to be calling into this function at the same time.

Parameters

<i>writer_filter_data</i>	<< <i>in</i> >> (p. 1013) A pointer to the state created using DDS_ContentFilter::writer_attach (p. 1577).
<i>cookie</i>	DDS_Cookie_t (p. 1580) to uniquely identify DDS_DataReader (p. 615) for which DDS_ContentFilter::writer_finalize (p. 1577) was called.

6.20.2.7 writer_attach

DDS_ContentFilterWriterAttachFunction `DDS_ContentFilter::writer_attach`

A writer-side filtering API to create some state that can facilitate filtering on the writer side.

This function is called to create some state required to perform filtering on the writer side using writer-side filtering APIs. This function will be called for every **DDS_DataWriter** (p. 485); it will be called only the first time the **DDS_DataWriter** (p. 485) matches a **DDS_DataReader** (p. 615) 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

<i>writer_filter_data</i>	<< out >> (p. 1013) A user-specified opaque pointer to some state created on the DDS_DataWriter (p. 485) that will help perform writer-side filtering efficiently.

6.20.2.8 writer_detach

DDS_ContentFilterWriterDetachFunction DDS_ContentFilter::writer_detach

A writer-side filtering API to clean up a previously created state using **DDS_ContentFilter::writer_attach** (p. 1577).

This function is called to delete any state created using the **DDS_ContentFilter::writer_attach** (p. 1577) function. This function will be called when the **DDS_DataWriter** (p. 485) is deleted.

Parameters

<i>writer_filter_data</i>	<< in >> (p. 1013) A pointer to the state created using DDS_ContentFilter::writer_attach (p. 1577).

6.20.2.9 writer_return_loan

DDS_ContentFilterWriterReturnLoanFunction DDS_ContentFilter::writer_return_loan

A writer-side filtering API to return the loan on the list of DataReaders returned by **DDS_ContentFilter::writer_evaluate** (p. 1576).

This function is called to return the loan on **DDS_CookieSeq** (p. 1581) returned by **DDS_ContentFilter::writer_↔return_loan** (p. 1578). It is possible for multiple threads to be calling into this function at the same time.

Parameters

<i>writer_filter_data</i>	<< in >> (p. 1013) A pointer to the state created using DDS_ContentFilter::writer_attach (p. 1577).
<i>cookies</i>	<< in >> (p. 1013) DDS_CookieSeq (p. 1581) for which the DDS_ContentFilter::writer_return_loan (p. 1578) was invoked.

6.20.2.10 filter_data

void* DDS_ContentFilter::filter_data

A place for filter implementors to keep a pointer to data that may be needed by their filter.

6.21 DDS_ContentFilterProperty_t Struct Reference

<<*extension*>> (p. 1012) Type used to provide all the required information to enable content filtering.

Data Fields

- 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.

6.21.1 Detailed Description

<<*extension*>> (p. 1012) Type used to provide all the required information to enable content filtering.

6.21.2 Field Documentation

6.21.2.1 content_filter_topic_name

```
char* DDS_ContentFilterProperty_t::content_filter_topic_name
```

Name of the Content-filtered Topic associated with the Reader.

6.21.2.2 related_topic_name

```
char* DDS_ContentFilterProperty_t::related_topic_name
```

Name of the Topic related to the Content-filtered Topic.

6.21.2.3 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).

6.21.2.4 filter_expression

```
char* DDS_ContentFilterProperty_t::filter_expression
```

The actual filter expression. Must be a valid expression for the filter class specified using filterClassName.

6.21.2.5 expression_parameters

```
struct DDS_StringSeq DDS_ContentFilterProperty_t::expression_parameters
```

Defines the value for each parameter in the filter expression.

6.22 DDS_Cookie_t Struct Reference

<<*extension*>> (p. 1012) Sequence of bytes.

Data Fields

- struct **DDS_OctetSeq** value
a sequence of octets

6.22.1 Detailed Description

<<*extension*>> (p. 1012) Sequence of bytes.

6.22.2 Field Documentation

6.22.2.1 value

```
struct DDS_OctetSeq DDS_Cookie_t::value
```

a sequence of octets

[default] Empty (zero-sized)

6.23 DDS_CookieSeq Struct Reference

Declares IDL sequence < DDS_Cookie_t (p. 1580) > .

6.23.1 Detailed Description

Declares IDL sequence < DDS_Cookie_t (p. 1580) > .

See also

FooSeq (p. 2113)

6.24 DDS_DatabaseQosPolicy Struct Reference

<<extension>> (p. 795) Various threads and resource limits settings used by RTI Connex to control its internal database.

Data Fields

- struct **DDS_ThreadSettings_t** thread
 <<extension>> (p. 795) Database thread settings.
- struct **DDS_Duration_t** shutdown_timeout
 [DEPRECATED] (p. 965) <<extension>> (p. 795) The maximum wait time during a shutdown.
- struct **DDS_Duration_t** cleanup_period
 [DEPRECATED] (p. 965) <<extension>> (p. 795) The period at which the database thread wakes up to remove deleted records.
- struct **DDS_Duration_t** shutdown_cleanup_period
 [DEPRECATED] (p. 965) <<extension>> (p. 795) The clean-up period used during database shut-down.
- **DDS_Long** initial_records
 <<extension>> (p. 795) The initial number of total records.
- **DDS_Long** max_skiplist_level
 [DEPRECATED] (p. 965) <<extension>> (p. 795) The maximum level of the skiplist.
- **DDS_Long** max_weak_references
 <<extension>> (p. 795) The maximum number of weak references.
- **DDS_Long** initial_weak_references
 <<extension>> (p. 795) The initial number of weak references.

6.24.1 Detailed Description

<<**extension**>> (p. 795) Various threads and resource limits settings used by RTI Connex 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 **DDS_DomainParticipant** (p. 77) that maintains this database is deleted, it shuts down this thread.

The Database QoS policy is used to configure how RTI Connex 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. 1583) and **DDS_DatabaseQosPolicy::shutdown_cleanup_period** (p. 1583) parameters to decrease the time it takes to delete a **DDS_DomainParticipant** (p. 77) when your application is shutting down.

The **DDS_DomainParticipantResourceLimitsQosPolicy** (p. 1731) controls the memory allocation for elements stored in the database.

This QoS policy is an extension to the DDS standard.

Entity:

DDS_DomainParticipant (p. 77)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) **NO** (p. 1245)

6.24.2 Field Documentation

6.24.2.1 thread

```
struct DDS_ThreadSettings_t DDS_DatabaseQosPolicy::thread
```

<<**extension**>> (p. 795) 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_STUDIO** (p. 1237)

Categories

Immutable (p. 914), **Concurrency** (p. 839)

6.24.2.2 shutdown_timeout

```
struct DDS_Duration_t DDS_DatabaseQosPolicy::shutdown_timeout
```

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) 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. 1208)]

Categories

Immutable (p. 914)

6.24.2.3 cleanup_period

```
struct DDS_Duration_t DDS_DatabaseQosPolicy::cleanup_period
```

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) The period at which the database thread wakes up to remove deleted records.

[default] 61 seconds

[range] [0,1 year]

Categories

Immutable (p. 914)

6.24.2.4 shutdown_cleanup_period

```
struct DDS_Duration_t DDS_DatabaseQosPolicy::shutdown_cleanup_period
```

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) 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. 914)

6.24.2.5 initial_records

```
DDS_Long DDS_DatabaseQosPolicy::initial_records
```

<<*extension*>> (p. 795) The initial number of total records.

[default] 1024

[range] [1,10 million]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.24.2.6 max_skiplist_level

```
DDS_Long DDS_DatabaseQosPolicy::max_skiplist_level
```

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) The maximum level of the skiplist.

The skiplist is used to keep records in the database. Usually, the search time is $\log_2(N)$, 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. 914), **Middleware Memory Management** (p. 878)

6.24.2.7 max_weak_references

DDS_Long DDS_DatabaseQosPolicy::max_weak_references

<<**extension**>> (p. 795) The maximum number of weak references.

A weak reference is an internal data structure that refers to a record within RTI Connex't internal database. This field configures the maximum number of such references that RTI Connex't may create.

The actual number of weak references is permitted to grow from an initial value (indicated by **DDS_DatabaseQosPolicy::initial_weak_references** (p. 1585)) to this maximum. To prevent RTI Connex't 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. 1326). 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 Connex't 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. 1326)

[range] [1, 100 million] or **DDS_LENGTH_UNLIMITED** (p. 1326), >= initial_weak_references

See also

DDS_DatabaseQosPolicy::initial_weak_references (p. 1585)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.24.2.8 initial_weak_references

DDS_Long DDS_DatabaseQosPolicy::initial_weak_references

<<**extension**>> (p. 795) The initial number of weak references.

See **DDS_DatabaseQosPolicy::max_weak_references** (p. 1584) 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. 1584) using RTI Connex't 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 Connex't 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. 1584)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.25 DDS_DataReaderCacheStatus Struct Reference

<<**extension**>> (p. 1012) The status of the reader's cache.

Data Fields

- **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. 1274).
- **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. 1274).
- **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. 2032).
- **DDS_LongLong expired_dropped_sample_count**
The number of samples expired by the DataReader due to **DDS_LifespanQosPolicy** (p. 1815) 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. 1293) 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. 711) that were dropped when removing an instance due to instance replacement. See **DDS_DataReaderResourceLimitsQosPolicy::instance_replacement** (p. 1635) 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. 715).
- **DDS_LongLong alive_instance_count_peak**
The highest value of **DDS_DataReaderCacheStatus::alive_instance_count** (p. 1590) 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. 715).
- **DDS_LongLong no_writers_instance_count_peak**

The highest value of **DDS_DataReaderCacheStatus::no_writers_instance_count** (p. 1591) 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. 715).

- **DDS_LongLong disposed_instance_count_peak**

The highest value of **DDS_DataReaderCacheStatus::disposed_instance_count** (p. 1591) 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. 1592) over the lifetime of the DataReader.

- **DDS_LongLong compressed_sample_count**

The number of received samples compressed by a DataWriter.

6.25.1 Detailed Description

<<**extension**>> (p. 1012) The status of the reader's cache.

Entity:

DDS_DataReader (p. 615)

6.25.2 Field Documentation

6.25.2.1 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.

6.25.2.2 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. 1304)

6.25.2.3 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. 1274).

When the DataReader is using **DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 1274):

- 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.

6.25.2.4 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. 1274).

When the DataReader is using **DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 1274): 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.

6.25.2.5 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.

6.25.2.6 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.

6.25.2.7 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. 2032).

When using **TIME_BASED_FILTER** (p. 1329) on the DataReader side, if the sample received by the DataReader does not pass the minimum_separation filter, it will be dropped.

6.25.2.8 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. 1815) or the autopurge sample delays.

- **DDS_LifespanQosPolicy** (p. 1815)
When a sample expires due to **DDS_LifespanQosPolicy** (p. 1815), the data is removed from the DataReader caches. This sample will be considered dropped if its **DDS_SampleStateKind** (p. 710) was **DDS_NOT_READ**↔**_SAMPLE_STATE** (p. 711).
- **DDS_ReaderDataLifecycleQosPolicy::autopurge_nowriter_samples_delay** (p. 1932)
When a sample expires due to **DDS_ReaderDataLifecycleQosPolicy::autopurge_nowriter_samples_delay** (p. 1932), this sample will be considered dropped if its **DDS_SampleStateKind** (p. 710) was **DDS_NOT_READ**↔**_SAMPLE_STATE** (p. 711).
- **DDS_ReaderDataLifecycleQosPolicy::autopurge_disposed_samples_delay** (p. 1932)
When a sample expires due to **DDS_ReaderDataLifecycleQosPolicy::autopurge_disposed_samples_delay** (p. 1932), this sample will be considered dropped if its **DDS_SampleStateKind** (p. 710) was **DDS_NOT_READ**↔**_SAMPLE_STATE** (p. 711).

6.25.2.9 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.

6.25.2.10 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. 1293) replacement.

When the number of samples for an instance in the queue reaches the **DDS_HistoryQosPolicy::depth** (p. 1807) 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. 710) was **DDS_NOT_READ_SAMPLE_STATE** (p. 711).

6.25.2.11 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. 1293) **DDS_HistoryQosPolicy** (p. 1805) QoS on the DataWriter or because the duration in **DDS_LifespanQosPolicy** (p. 1815) 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. 1901) to true). Note: Historical data with removed batch samples written before the DataReader joined the DDS domain will also be included in the count.

6.25.2.12 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. 711) that were dropped when removing an instance due to instance replacement. See **DDS_DataReaderResourceLimitsQosPolicy::instance_replacement** (p. 1635) for more details about when instances are replaced.

6.25.2.13 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. 715).

6.25.2.14 alive_instance_count_peak

DDS_LongLong DDS_DataReaderCacheStatus::alive_instance_count_peak

The highest value of **DDS_DataReaderCacheStatus::alive_instance_count** (p. 1590) over the lifetime of the DataReader.

See also

DDS_DataReaderCacheStatus::alive_instance_count (p. 1590)

6.25.2.15 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. 715).

6.25.2.16 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. 1591) over the lifetime of the DataReader.

See also

DDS_DataReaderCacheStatus::no_writers_instance_count (p. 1591)

6.25.2.17 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. 715).

6.25.2.18 disposed_instance_count_peak

DDS_LongLong DDS_DataReaderCacheStatus::disposed_instance_count_peak

The highest value of **DDS_DataReaderCacheStatus::disposed_instance_count** (p. 1591) over the lifetime of the DataReader.

See also

DDS_DataReaderCacheStatus::disposed_instance_count (p. 1591)

6.25.2.19 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. 1633) is true, the DataReader will keep up to a maximum of **DDS_DataReaderResourceLimitsQosPolicy::max_total_instances** (p. 1629) detached instances in its queue. For a more in-depth description of detached instances, refer to **DDS_DataReaderResourceLimitsQosPolicy::max_total_instances** (p. 1629).

6.25.2.20 detached_instance_count_peak

DDS_LongLong DDS_DataReaderCacheStatus::detached_instance_count_peak

The highest value of **DDS_DataReaderCacheStatus::detached_instance_count** (p. 1592) over the lifetime of the DataReader.

See also

DDS_DataReaderCacheStatus::detached_instance_count (p. 1592)

6.25.2.21 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.

6.26 DDS_DataReaderListener Struct Reference

<<*interface*>> (p. 1013) **DDS_Listener** (p. 1816) for reader status.

Data Fields

- struct **DDS_Listener as_listener**
*The superclass instance of this **DDS_DataReaderListener** (p. 1593).*
- **DDS_DataReaderListener_RequestedDeadlineMissedCallback on_requested_deadline_missed**
*Handles the **DDS_REQUESTED_DEADLINE_MISSED_STATUS** (p. 1229) communication status.*
- **DDS_DataReaderListener_RequestedIncompatibleQosCallback on_requested_incompatible_qos**
*Handles the **DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 1229) communication status.*
- **DDS_DataReaderListener_SampleRejectedCallback on_sample_rejected**
*Handles the **DDS_SAMPLE_REJECTED_STATUS** (p. 1230) communication status.*
- **DDS_DataReaderListener_LivelinessChangedCallback on_liveliness_changed**
*Handles the **DDS_LIVELINESS_CHANGED_STATUS** (p. 1231) communication status.*
- **DDS_DataReaderListener_DataAvailableCallback on_data_available**
*Handle the **DDS_DATA_AVAILABLE_STATUS** (p. 1230) communication status.*
- **DDS_DataReaderListener_SubscriptionMatchedCallback on_subscription_matched**
*Handles the **DDS_SUBSCRIPTION_MATCHED_STATUS** (p. 1232) communication status.*
- **DDS_DataReaderListener_SampleLostCallback on_sample_lost**
*Handles the **DDS_SAMPLE_LOST_STATUS** (p. 1230) communication status.*

6.26.1 Detailed Description

<<*interface*>> (p. 1013) **DDS_Listener** (p. 1816) for reader status.

Entity:

DDS_DataReader (p. 615)

Status:

DDS_DATA_AVAILABLE_STATUS (p. 1230);
DDS_LIVELINESS_CHANGED_STATUS (p. 1231), **DDS_LivelinessChangedStatus** (p. 1820);
DDS_REQUESTED_DEADLINE_MISSED_STATUS (p. 1229), **DDS_RequestedDeadlineMissedStatus** (p. 1946);
DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS (p. 1229), **DDS_RequestedIncompatibleQosStatus** (p. 1947);
DDS_SAMPLE_LOST_STATUS (p. 1230), **DDS_SampleLostStatus** (p. 1996);
DDS_SAMPLE_REJECTED_STATUS (p. 1230), **DDS_SampleRejectedStatus** (p. 1997);
DDS_SUBSCRIPTION_MATCHED_STATUS (p. 1232), **DDS_SubscriptionMatchedStatus** (p. 2022);

See also

Status Kinds (p. 1222)

Operations Allowed in Listener Callbacks (p. 1818)

Examples

HelloWorld_subscriber.c.

6.26.2 Field Documentation

6.26.2.1 as_listener

```
struct DDS_Listener DDS_DataReaderListener::as_listener
```

The superclass instance of this **DDS_DataReaderListener** (p. 1593).

6.26.2.2 on_requested_deadline_missed

```
DDS_DataReaderListener_RequestedDeadlineMissedCallback DDS_DataReaderListener::on_requested_↔  
deadline_missed
```

Handles the **DDS_REQUESTED_DEADLINE_MISSED_STATUS** (p. 1229) communication status.

Examples

```
HelloWorld_subscriber.c.
```

6.26.2.3 on_requested_incompatible_qos

```
DDS_DataReaderListener_RequestedIncompatibleQosCallback DDS_DataReaderListener::on_requested_↔  
incompatible_qos
```

Handles the **DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 1229) communication status.

Examples

```
HelloWorld_subscriber.c.
```

6.26.2.4 on_sample_rejected

```
DDS_DataReaderListener_SampleRejectedCallback DDS_DataReaderListener::on_sample_rejected
```

Handles the **DDS_SAMPLE_REJECTED_STATUS** (p. 1230) communication status.

Examples

```
HelloWorld_subscriber.c.
```

6.26.2.5 on_liveliness_changed

DDS_DataReaderListener_LivelinessChangedCallback DDS_DataReaderListener::on_liveliness_changed

Handles the **DDS_LIVELINESS_CHANGED_STATUS** (p. 1231) communication status.

Examples

HelloWorld_subscriber.c.

6.26.2.6 on_data_available

DDS_DataReaderListener_DataAvailableCallback DDS_DataReaderListener::on_data_available

Handle the **DDS_DATA_AVAILABLE_STATUS** (p. 1230) communication status.

Examples

HelloWorld_subscriber.c.

6.26.2.7 on_subscription_matched

DDS_DataReaderListener_SubscriptionMatchedCallback DDS_DataReaderListener::on_subscription_↔
matched

Handles the **DDS_SUBSCRIPTION_MATCHED_STATUS** (p. 1232) communication status.

Examples

HelloWorld_subscriber.c.

6.26.2.8 on_sample_lost

DDS_DataReaderListener_SampleLostCallback DDS_DataReaderListener::on_sample_lost

Handles the **DDS_SAMPLE_LOST_STATUS** (p. 1230) communication status.

Examples

HelloWorld_subscriber.c.

6.27 DDS_DataReaderProtocolQosPolicy Struct Reference

<<*extension*>> (p. 795) Along with **DDS_WireProtocolQosPolicy** (p. 2093) and **DDS_DataWriterProtocolQosPolicy** (p. 1649), this QoS policy configures the DDS on-the-network protocol (RTPS).

Data Fields

- struct **DDS_GUID_t virtual_guid**
 <<*extension*>> (p. 795) *The virtual GUID (Global Unique Identifier).*
- **DDS_UnsignedLong rtps_object_id**
 <<*extension*>> (p. 795) *The RTPS Object ID.*
- **DDS_Boolean expects_inline_qos**
 [DEPRECATED] (p. 965) <<*extension*>> (p. 795) *Specifies whether this DataReader expects inline QoS with every sample.*
- **DDS_Boolean disable_positive_acks**
 <<*extension*>> (p. 795) *Whether the reader sends positive acknowledgements to writers.*
- **DDS_Boolean propagate_dispose_of_unregistered_instances**
 <<*extension*>> (p. 795) *Indicates whether or not an instance can move to the **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 715) state without being in the **DDS_ALIVE_INSTANCE_STATE** (p. 715) state.*
- **DDS_Boolean propagate_unregister_of_disposed_instances**
 <<*extension*>> (p. 795) *Indicates whether or not an instance can move to the **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 715) state directly from the **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 715).*
- struct **DDS_RtpsReliableReaderProtocol_t rtps_reliable_reader**
 <<*extension*>> (p. 795) *RTPS protocol-related configuration settings for the RTPS reliable reader associated with a **DDS_DataReader** (p. 615). This parameter only has effect if the reader is configured with **DDS_RELIABLE_RELIABILITY_QOS** (p. 1323) **DDS_ReliabilityQosPolicyKind** (p. 1322).*

6.27.1 Detailed Description

<<*extension*>> (p. 795) Along with **DDS_WireProtocolQosPolicy** (p. 2093) and **DDS_DataWriterProtocolQosPolicy** (p. 1649), 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. 1596) 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 **DDS_DataWriter** (p. 485) and **DDS_DataReader** (p. 615) 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. 1936) for more information on the per-DataReader/DataWriter reliability configuration. **DDS_HistoryQosPolicy** (p. 1805) and **DDS_ResourceLimitsQosPolicy** (p. 1949) also play an important role in the DDS reliable protocol.

This QoS policy is an extension to the DDS standard.

Entity:

DDS_DataReader (p. 615)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

6.27.2 Field Documentation

6.27.2.1 virtual_guid

```
struct DDS_GUID_t DDS_DataReaderProtocolQosPolicy::virtual_guid
```

<<**extension**>> (p. 795) The virtual GUID (Global Unique Identifier).

The virtual GUID is used to uniquely identify different incarnations of the same **DDS_DataReader** (p. 615).

The association between a **DDS_DataReader** (p. 615) and its persisted state is done using the virtual GUID.

[default] **DDS_GUID_AUTO** (p. 1213)

Categories

Immutable (p. 914), **Discovery Metadata** (p. 958)

6.27.2.2 rtps_object_id

```
DDS_UnsignedLong DDS_DataReaderProtocolQosPolicy::rtps_object_id
```

<<**extension**>> (p. 795) 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 Connexx 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 Connexx. In those cases, the recommendation is not to use automatic object ID assignment.

[default] **DDS_RTPS_AUTO_ID** (p. ??)

[range] [0,0x00ffffff]

Categories

Immutable (p. 914)

6.27.2.3 expects_inline_qos

DDS_Boolean DDS_DataReaderProtocolQosPolicy::expects_inline_qos

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) 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. 1201) (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 **DDS_DataReader** (p. 615) nominally relies on Discovery to propagate QoS on a matched **DDS_↔DataWriter** (p. 485).

Alternatively, a **DDS_DataReader** (p. 615) may get information on a matched **DDS_DataWriter** (p. 485) through QoS sent inline with a sample.

Asserting **DDS_DataReaderProtocolQosPolicy::expects_inline_qos** (p. 1597) indicates to a matching **DDS_Data↔Writer** (p. 485) that this **DDS_DataReader** (p. 615) expects to receive inline QoS with every sample. The complete set of inline QoS that a **DDS_DataWriter** (p. 485) may send inline is specified by the Real-Time Publish-Subscribe (RTPS) Wire Interoperability Protocol.

Because RTI Connext **DDS_DataWriter** (p. 485) and **DDS_DataReader** (p. 615) cache Discovery information, inline QoS are largely redundant and thus unnecessary. Only for other stateless implementations whose **DDS_DataReader** (p. 615) 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. 1201)

Categories

Immutable (p. 914), **Discovery Metadata** (p. 958)

6.27.2.4 disable_positive_acks

DDS_Boolean DDS_DataReaderProtocolQosPolicy::disable_positive_acks

<<*extension*>> (p. 795) Whether the reader sends positive acknowledgements to writers.

If set to **DDS_BOOLEAN_TRUE** (p. 1201), 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. 1201) (the default), the reader will send ACKs to writers that expect ACKs (**DDS_DataWriterProtocolQosPolicy::disable_positive_acks** (p. 1651) = **DDS_BOOLEAN_FALSE** (p. 1201)) and it will not send ACKs to writers that disable ACKs (**DDS_DataWriterProtocolQosPolicy::disable_positive_acks** (p. 1651) = **DDS_BOOLEAN_TRUE** (p. 1201))

[default] **DDS_BOOLEAN_FALSE** (p. 1201)

Categories

Immutable (p. 914), **Discovery Metadata** (p. 958), **Reliability** (p. 895)

6.27.2.5 propagate_dispose_of_unregistered_instances

DDS_Boolean DDS_DataReaderProtocolQosPolicy::propagate_dispose_of_unregistered_instances

<<*extension*>> (p. 795) Indicates whether or not an instance can move to the **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 715) state without being in the **DDS_ALIVE_INSTANCE_STATE** (p. 715) state.

This field only applies to keyed readers.

When the field is set to **DDS_BOOLEAN_TRUE** (p. 1201), 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. 647), this option should be used in combination with setting **DDS_DataWriterProtocolQosPolicy::serialize_key_with_dispose** (p. 1652) on the DataWriter to **DDS_BOOLEAN_TRUE** (p. 1201).

[default] **DDS_BOOLEAN_FALSE** (p. 1201)

Categories

Immutable (p. 914), **Instance Lifecycle** (p. 867)

6.27.2.6 propagate_unregister_of_disposed_instances

DDS_Boolean DDS_DataReaderProtocolQosPolicy::propagate_unregister_of_disposed_instances

<<*extension*>> (p. 795) Indicates whether or not an instance can move to the **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 715) state directly from the **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 715).

This field only applies to keyed readers.

When the field is set to **DDS_BOOLEAN_TRUE** (p. 1201), the DataReader will receive unregister notifications even if the instance is not alive.

[default] **DDS_BOOLEAN_FALSE** (p. 1201)

Categories

Immutable (p. 914), **Instance Lifecycle** (p. 867)

6.27.2.7 rtps_reliable_reader

```
struct DDS_RtpsReliableReaderProtocol_t DDS_DataReaderProtocolQosPolicy::rtps_reliable_reader
```

<<**extension**>> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable reader associated with a **DDS_DataReader** (p. 615). This parameter only has effect if the reader is configured with **DDS_RELIABLE_↔RELIABILITY_QOS** (p. 1323) **DDS_ReliabilityQosPolicyKind** (p. 1322).

For details, refer to the **DDS_RtpsReliableReaderProtocol_t** (p. 1954)

[default] See **DDS_RtpsReliableReaderProtocol_t** (p. 1954)

Categories

Immutable (p. 914), **Reliability** (p. 895)

6.28 DDS_DataReaderProtocolStatus Struct Reference

<<**extension**>> (p. 1012) The status of a reader's internal protocol related metrics, like the number of samples received, filtered, rejected; and status of wire protocol traffic.

Data Fields

- **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. 1602) 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. 1603) 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. 1604) 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. 1604) since the last time the status was read.*
- **DDS_LongLong filtered_sample_count**
*[DEPRECATED]. See: **DDS_DataReaderCacheStatus::time_based_filter_dropped_sample_count** (p. 1588) **DDS_↔DataReaderCacheStatus::content_filter_dropped_sample_count** (p. 1588)*
- **DDS_LongLong filtered_sample_count_change**

[DEPRECATED]. See: `DDS_DataReaderCacheStatus::time_based_filter_dropped_sample_count` (p. 1588) `DDS_DataReaderCacheStatus::content_filter_dropped_sample_count` (p. 1588)

- **DDS_LongLong filtered_sample_bytes**

[DEPRECATED]. See: `DDS_DataReaderCacheStatus::time_based_filter_dropped_sample_count` (p. 1588) `DDS_DataReaderCacheStatus::content_filter_dropped_sample_count` (p. 1588)

- **DDS_LongLong filtered_sample_bytes_change**

[DEPRECATED]. See: `DDS_DataReaderCacheStatus::time_based_filter_dropped_sample_count` (p. 1588) `DDS_DataReaderCacheStatus::content_filter_dropped_sample_count` (p. 1588)

- **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. 1605) 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. 1606) 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. 1606) 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. 1606) 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. 1607) 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. 1607) 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. 1607) 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. 1608) 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. 1608) 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.

6.28.1 Detailed Description

<<*extension*>> (p. 1012) 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:

DDS_DataReader (p. 615)

6.28.2 Field Documentation

6.28.2.1 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. 1600) 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.1600) is obtained using the **DDS_DataReader_get_datareader_protocol_status** (p. 686) operation then it will count samples from any DataWriter. If the **DataReaderProtocolStatus** is obtained using the **DDS_DataReader_get_matched_publication_datareader_protocol_status** (p. 686) 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. 1604).

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. 1604).

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.

6.28.2.2 received_sample_count_change

DDS_LongLong DDS_DataReaderProtocolStatus::received_sample_count_change

The change in **DDS_DataReaderProtocolStatus::received_sample_count** (p. 1602) since the last time the status was read.

See also

DDS_DataReaderProtocolStatus::received_sample_count (p. 1602)

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.

6.28.2.3 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. 1602)

Note that when data is fragmented, this statistic is updated upon the receipt of each fragment, not when a sample is reassembled.

6.28.2.4 received_sample_bytes_change

DDS_LongLong DDS_DataReaderProtocolStatus::received_sample_bytes_change

The change in **DDS_DataReaderProtocolStatus::received_sample_bytes** (p. 1603) since the last time the status was read.

See also

DDS_DataReaderProtocolStatus::received_sample_count_change (p. 1603)

Note that when data is fragmented, this statistic is updated upon the receipt of each fragment, not when a sample is reassembled.

6.28.2.5 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.

6.28.2.6 duplicate_sample_count_change

DDS_LongLong DDS_DataReaderProtocolStatus::duplicate_sample_count_change

The change in **DDS_DataReaderProtocolStatus::duplicate_sample_count** (p. 1604) 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.

6.28.2.7 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.

6.28.2.8 duplicate_sample_bytes_change

DDS_LongLong DDS_DataReaderProtocolStatus::duplicate_sample_bytes_change

The change in **DDS_DataReaderProtocolStatus::duplicate_sample_bytes** (p. 1604) 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.

6.28.2.9 filtered_sample_count

DDS_LongLong DDS_DataReaderProtocolStatus::filtered_sample_count

[DEPRECATED]. See: **DDS_DataReaderCacheStatus::time_based_filter_dropped_sample_count** (p. 1588)
DDS_DataReaderCacheStatus::content_filter_dropped_sample_count (p. 1588)

6.28.2.10 filtered_sample_count_change

DDS_LongLong DDS_DataReaderProtocolStatus::filtered_sample_count_change

[DEPRECATED]. See: **DDS_DataReaderCacheStatus::time_based_filter_dropped_sample_count** (p. 1588)
DDS_DataReaderCacheStatus::content_filter_dropped_sample_count (p. 1588)

6.28.2.11 filtered_sample_bytes

DDS_LongLong DDS_DataReaderProtocolStatus::filtered_sample_bytes

[DEPRECATED]. See: **DDS_DataReaderCacheStatus::time_based_filter_dropped_sample_count** (p. 1588)
DDS_DataReaderCacheStatus::content_filter_dropped_sample_count (p. 1588)

6.28.2.12 filtered_sample_bytes_change

DDS_LongLong DDS_DataReaderProtocolStatus::filtered_sample_bytes_change

[DEPRECATED]. See: **DDS_DataReaderCacheStatus::time_based_filter_dropped_sample_count** (p. 1588)
DDS_DataReaderCacheStatus::content_filter_dropped_sample_count (p. 1588)

6.28.2.13 received_heartbeat_count

DDS_LongLong DDS_DataReaderProtocolStatus::received_heartbeat_count

The number of Heartbeats from a remote DataWriter received by a local DataReader.

6.28.2.14 received_heartbeat_count_change

DDS_LongLong DDS_DataReaderProtocolStatus::received_heartbeat_count_change

The change in **DDS_DataReaderProtocolStatus::received_heartbeat_count** (p. 1605) since the last time the status was read.

6.28.2.15 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.

6.28.2.16 received_heartbeat_bytes_change

DDS_LongLong DDS_DataReaderProtocolStatus::received_heartbeat_bytes_change

The change in **DDS_DataReaderProtocolStatus::received_heartbeat_bytes** (p. 1606) since the last time the status was read.

6.28.2.17 sent_ack_count

DDS_LongLong DDS_DataReaderProtocolStatus::sent_ack_count

The number of ACKs sent from a local DataReader to a matching remote DataWriter.

6.28.2.18 sent_ack_count_change

DDS_LongLong DDS_DataReaderProtocolStatus::sent_ack_count_change

The change in **DDS_DataReaderProtocolStatus::sent_ack_count** (p. 1606) since the last time the status was read.

6.28.2.19 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.

6.28.2.20 sent_ack_bytes_change

DDS_LongLong DDS_DataReaderProtocolStatus::sent_ack_bytes_change

The change in **DDS_DataReaderProtocolStatus::sent_ack_bytes** (p. 1606) since the last time the status was read.

6.28.2.21 sent_nack_count

DDS_LongLong DDS_DataReaderProtocolStatus::sent_nack_count

The number of NACKs sent from a local DataReader to a matching remote DataWriter.

6.28.2.22 sent_nack_count_change

DDS_LongLong DDS_DataReaderProtocolStatus::sent_nack_count_change

The change in **DDS_DataReaderProtocolStatus::sent_nack_count** (p. 1607) since the last time the status was read.

6.28.2.23 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.

6.28.2.24 sent_nack_bytes_change

DDS_LongLong DDS_DataReaderProtocolStatus::sent_nack_bytes_change

The change in **DDS_DataReaderProtocolStatus::sent_nack_bytes** (p. 1607) since the last time the status was read.

6.28.2.25 received_gap_count

DDS_LongLong DDS_DataReaderProtocolStatus::received_gap_count

The number of GAPS received from remote DataWriter to this DataReader.

6.28.2.26 received_gap_count_change

DDS_LongLong DDS_DataReaderProtocolStatus::received_gap_count_change

The change in **DDS_DataReaderProtocolStatus::received_gap_count** (p. 1607) since the last time the status was read.

6.28.2.27 received_gap_bytes

DDS_LongLong DDS_DataReaderProtocolStatus::received_gap_bytes

The number of bytes of GAPS received from remote DataWriter to this DataReader.

6.28.2.28 received_gap_bytes_change

DDS_LongLong DDS_DataReaderProtocolStatus::received_gap_bytes_change

The change in **DDS_DataReaderProtocolStatus::received_gap_bytes** (p. 1608) since the last time the status was read.

6.28.2.29 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. 1323).

6.28.2.30 rejected_sample_count_change

DDS_LongLong DDS_DataReaderProtocolStatus::rejected_sample_count_change

The change in **DDS_DataReaderProtocolStatus::rejected_sample_count** (p. 1608) since the last time the status was read.

This counter will always be 0 when using **DDS_BEST_EFFORT_RELIABILITY_QOS** (p. 1323).

6.28.2.31 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.

6.28.2.32 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.

6.28.2.33 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.

6.28.2.34 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.

6.28.2.35 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. 1323); 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. 1957). When the received window is full any out-of-order sample received will be dropped.

6.28.2.36 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.

6.28.2.37 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.

6.28.2.38 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.

6.28.2.39 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.

6.28.2.40 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.

6.29 DDS_DataReaderQos Struct Reference

QoS policies supported by a **DDS_DataReader** (p. 615) entity.

Data Fields

- struct **DDS_DurabilityQosPolicy** durability
*Durability policy, **DURABILITY** (p. 1285).*
- struct **DDS_DeadlineQosPolicy** deadline
*<<basic>> (p. 783) Deadline policy, **DEADLINE** (p. 1272).*
- struct **DDS_LatencyBudgetQosPolicy** latency_budget
*Latency budget policy, **LATENCY_BUDGET** (p. 1294).*
- struct **DDS_LivelinessQosPolicy** liveliness
*Liveliness policy, **LIVELINESS** (p. 1296).*
- struct **DDS_ReliabilityQosPolicy** reliability
*Reliability policy, **RELIABILITY** (p. 1321).*
- struct **DDS_DestinationOrderQosPolicy** destination_order
*Destination order policy, **DESTINATION_ORDER** (p. 1273).*
- struct **DDS_HistoryQosPolicy** history
*<<basic>> (p. 783) History policy, **HISTORY** (p. 1292).*
- struct **DDS_ResourceLimitsQosPolicy** resource_limits
*Resource limits policy, **RESOURCE_LIMITS** (p. 1325).*
- struct **DDS_UserDataQosPolicy** user_data
*User data policy, **USER_DATA** (p. 1343).*
- struct **DDS_OwnershipQosPolicy** ownership
*<<basic>> (p. 783) Ownership policy, **OWNERSHIP** (p. 1301).*
- struct **DDS_TimeBasedFilterQosPolicy** time_based_filter
*<<basic>> (p. 783) Time-based filter policy, **TIME_BASED_FILTER** (p. 1329).*
- struct **DDS_ReaderDataLifecycleQosPolicy** reader_data_lifecycle
*Reader data lifecycle policy, **READER_DATA_LIFECYCLE** (p. 1320).*
- struct **DDS_DataRepresentationQosPolicy** representation
*Data representation policy, **DATA_REPRESENTATION** (p. 1256).*
- struct **DDS_TypeConsistencyEnforcementQosPolicy** type_consistency
*Type consistency enforcement policy, **TYPE_CONSISTENCY_ENFORCEMENT** (p. 1339).*
- **DDS_DataTagQosPolicy** data_tags

*DataTag policy, **DATA_TAG** (p. 1263).*

- struct **DDS_DataReaderResourceLimitsQosPolicy** **reader_resource_limits**
 <<*extension*>> (p. 1012) *DDS_DataReader* (p. 615) *resource limits policy, **DATA_READER_RESOURCE_LIMITS** (p. 1253). This policy is an extension to the DDS standard.*
- struct **DDS_DataReaderProtocolQosPolicy** **protocol**
 <<*extension*>> (p. 1012) *DDS_DataReader* (p. 615) *protocol policy, **DATA_READER_PROTOCOL** (p. 1253)*
- struct **DDS_TransportSelectionQosPolicy** **transport_selection**
 <<*extension*>> (p. 1012) *Transport selection policy, **TRANSPORT_SELECTION** (p. 1338).*
- struct **DDS_TransportUnicastQosPolicy** **unicast**
 <<*extension*>> (p. 1012) *Unicast transport policy, **TRANSPORT_UNICAST** (p. 1339).*
- struct **DDS_TransportMulticastQosPolicy** **multicast**
 <<*extension*>> (p. 1012) *Multicast transport policy, **TRANSPORT_MULTICAST** (p. 1335).*
- struct **DDS_PropertyQosPolicy** **property**
 <<*basic*>> (p. 783) <<*extension*>> (p. 1012) *Property policy, **PROPERTY** (p. 1306). See also *Property Reference Guide*.*
- struct **DDS_ServiceQosPolicy** **service**
 <<*extension*>> (p. 1012) *Service policy, **SERVICE** (p. 1326).*
- struct **DDS_AvailabilityQosPolicy** **availability**
 <<*extension*>> (p. 1012) *Availability policy, **AVAILABILITY** (p. 1250).*
- struct **DDS_EntityNameQosPolicy** **subscription_name**
 <<*extension*>> (p. 1012) *EntityName policy, **ENTITY_NAME** (p. 1290).*
- struct **DDS_TransportPriorityQosPolicy** **transport_priority**
*Transport priority policy, **TRANSPORT_PRIORITY** (p. 1337).*
- struct **DDS_TypeSupportQosPolicy** **type_support**
 <<*extension*>> (p. 1012) *type support data, **TYPESUPPORT** (p. 1341).*

6.29.1 Detailed Description

QoS policies supported by a **DDS_DataReader** (p. 615) entity.

You must set certain members in a consistent manner:

DDS_DataReaderQos::deadline (p. 1613) .period >= **DDS_DataReaderQos::time_based_filter** (p. 1615) .minimum_separation

DDS_DataReaderQos::history (p. 1614) .depth <= **DDS_DataReaderQos::resource_limits** (p. 1614) .max_↵ samples_per_instance

DDS_DataReaderQos::resource_limits (p. 1614) .max_samples_per_instance <= **DDS_DataReaderQos::↵ resource_limits** (p. 1614) .max_samples **DDS_DataReaderQos::resource_limits** (p. 1614) .initial_samples <= **DDS_DataReaderQos::resource_limits** (p. 1614) .max_samples

DDS_DataReaderQos::resource_limits (p. 1614) .initial_instances <= **DDS_DataReaderQos::resource_limits** (p. 1614) .max_instances

DDS_DataReaderQos::reader_resource_limits (p. 1615) .initial_remote_writers_per_instance <= **DDS_Data↵ ReaderQos::reader_resource_limits** (p. 1615) .max_remote_writers_per_instance

DDS_DataReaderQos::reader_resource_limits (p. 1615) .initial_infos <= **DDS_DataReaderQos::reader_resource_limits** (p. 1615) .max_infos

DDS_DataReaderQos::reader_resource_limits (p. 1615) .max_remote_writers_per_instance <= **DDS_DataReaderQos::reader_resource_limits** (p. 1615) .max_remote_writers

DDS_DataReaderQos::reader_resource_limits (p. 1615) .max_samples_per_remote_writer <= **DDS_DataReaderQos::resource_limits** (p. 1614) .max_samples

length of **DDS_DataReaderQos::user_data** (p. 1614) .value <= **DDS_DomainParticipantQos::resource_limits** (p. 1729) .reader_user_data_max_length

If any of the above are not true, **DDS_DataReader_set_qos** (p. 687) and **DDS_DataReader_set_qos_with_profile** (p. 687) will fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222) and **DDS_Subscriber_create_datareader** (p. 581) will return NULL.

6.29.2 Field Documentation

6.29.2.1 durability

```
struct DDS_DurabilityQosPolicy DDS_DataReaderQos::durability
```

Durability policy, **DURABILITY** (p. 1285).

6.29.2.2 deadline

```
struct DDS_DeadlineQosPolicy DDS_DataReaderQos::deadline
```

<<**basic**>> (p. 783) Deadline policy, **DEADLINE** (p. 1272).

6.29.2.3 latency_budget

```
struct DDS_LatencyBudgetQosPolicy DDS_DataReaderQos::latency_budget
```

Latency budget policy, **LATENCY_BUDGET** (p. 1294).

6.29.2.4 liveliness

```
struct DDS_LivelinessQosPolicy DDS_DataReaderQos::liveliness
```

Liveliness policy, **LIVELINESS** (p. 1296).

6.29.2.5 reliability

```
struct DDS_ReliabilityQosPolicy DDS_DataReaderQos::reliability
```

Reliability policy, **RELIABILITY** (p. 1321).

6.29.2.6 destination_order

```
struct DDS_DestinationOrderQosPolicy DDS_DataReaderQos::destination_order
```

Destination order policy, **DESTINATION_ORDER** (p. 1273).

6.29.2.7 history

```
struct DDS_HistoryQosPolicy DDS_DataReaderQos::history
```

<<*basic*>> (p. 783) History policy, **HISTORY** (p. 1292).

6.29.2.8 resource_limits

```
struct DDS_ResourceLimitsQosPolicy DDS_DataReaderQos::resource_limits
```

Resource limits policy, **RESOURCE_LIMITS** (p. 1325).

6.29.2.9 user_data

```
struct DDS_UserDataQosPolicy DDS_DataReaderQos::user_data
```

User data policy, **USER_DATA** (p. 1343).

6.29.2.10 ownership

```
struct DDS_OwnershipQosPolicy DDS_DataReaderQos::ownership
```

<<**basic**>> (p. 783) Ownership policy, **OWNERSHIP** (p. 1301).

6.29.2.11 time_based_filter

```
struct DDS_TimeBasedFilterQosPolicy DDS_DataReaderQos::time_based_filter
```

<<**basic**>> (p. 783) Time-based filter policy, **TIME_BASED_FILTER** (p. 1329)

6.29.2.12 reader_data_lifecycle

```
struct DDS_ReaderDataLifecycleQosPolicy DDS_DataReaderQos::reader_data_lifecycle
```

Reader data lifecycle policy, **READER_DATA_LIFECYCLE** (p. 1320).

6.29.2.13 representation

```
struct DDS_DataRepresentationQosPolicy DDS_DataReaderQos::representation
```

Data representation policy, **DATA_REPRESENTATION** (p. 1256).

6.29.2.14 type_consistency

```
struct DDS_TypeConsistencyEnforcementQosPolicy DDS_DataReaderQos::type_consistency
```

Type consistency enforcement policy, **TYPE_CONSISTENCY_ENFORCEMENT** (p. 1339).

6.29.2.15 data_tags

```
DDS_DataTagQosPolicy DDS_DataReaderQos::data_tags
```

DataTag policy, **DATA_TAG** (p. 1263).

6.29.2.16 reader_resource_limits

```
struct DDS_DataReaderResourceLimitsQosPolicy DDS_DataReaderQos::reader_resource_limits
```

<<*extension*>> (p. 1012) **DDS_DataReader** (p. 615) resource limits policy, **DATA_READER_RESOURCE_LIMITS** (p. 1253). This policy is an extension to the DDS standard.

6.29.2.17 protocol

```
struct DDS_DataReaderProtocolQosPolicy DDS_DataReaderQos::protocol
```

<<*extension*>> (p. 1012) **DDS_DataReader** (p. 615) protocol policy, **DATA_READER_PROTOCOL** (p. 1253)

6.29.2.18 transport_selection

```
struct DDS_TransportSelectionQosPolicy DDS_DataReaderQos::transport_selection
```

<<*extension*>> (p. 1012) Transport selection policy, **TRANSPORT_SELECTION** (p. 1338).

Specifies the transports available for use by the **DDS_DataReader** (p. 615).

6.29.2.19 unicast

```
struct DDS_TransportUnicastQosPolicy DDS_DataReaderQos::unicast
```

<<*extension*>> (p. 1012) Unicast transport policy, **TRANSPORT_UNICAST** (p. 1339).

Specifies the unicast transport interfaces and ports on which **messages** can be received.

The unicast interfaces are used to receive messages from **DDS_DataWriter** (p. 485) entities in the domain.

6.29.2.20 multicast

```
struct DDS_TransportMulticastQosPolicy DDS_DataReaderQos::multicast
```

<<*extension*>> (p. 1012) Multicast transport policy, **TRANSPORT_MULTICAST** (p. 1335).

Specifies the multicast group addresses and ports on which **messages** can be received.

The multicast addresses are used to receive messages from **DDS_DataWriter** (p. 485) entities in the domain.

6.29.2.21 property

```
struct DDS_PropertyQosPolicy DDS_DataReaderQos::property
```

<<**basic**>> (p. 783) <<**extension**>> (p. 1012) Property policy, **PROPERTY** (p.1306). See also Property Reference Guide.

6.29.2.22 service

```
struct DDS_ServiceQosPolicy DDS_DataReaderQos::service
```

<<**extension**>> (p. 1012) Service policy, **SERVICE** (p. 1326).

6.29.2.23 availability

```
struct DDS_AvailabilityQosPolicy DDS_DataReaderQos::availability
```

<<**extension**>> (p. 1012) Availability policy, **AVAILABILITY** (p. 1250).

6.29.2.24 subscription_name

```
struct DDS_EntityNameQosPolicy DDS_DataReaderQos::subscription_name
```

<<**extension**>> (p. 1012) EntityName policy, **ENTITY_NAME** (p. 1290).

6.29.2.25 transport_priority

```
struct DDS_TransportPriorityQosPolicy DDS_DataReaderQos::transport_priority
```

Transport priority policy, **TRANSPORT_PRIORITY** (p. 1337).

6.29.2.26 type_support

```
struct DDS_TypeSupportQosPolicy DDS_DataReaderQos::type_support
```

<<**extension**>> (p. 1012) type support data, **TYPESUPPORT** (p. 1341).

Optional value that is passed to a type plugin's on_endpoint_attached and deserialization functions.

6.30 DDS_DataReaderResourceLimitsInstanceReplacementSettings Struct Reference

Instance replacement kind applied to each instance state.

Data Fields

- **DDS_DataReaderInstanceRemovalKind alive_instance_removal**
*Removal kind applied to alive (**DDS_ALIVE_INSTANCE_STATE** (p. 715)) instances.*
- **DDS_DataReaderInstanceRemovalKind disposed_instance_removal**
*Removal kind applied to disposed (**DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 715)) instances.*
- **DDS_DataReaderInstanceRemovalKind no_writers_instance_removal**
*Removal kind applied to fully-unregistered (**DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 715)) instances.*

6.30.1 Detailed Description

Instance replacement kind applied to each instance state.

[default]

- **DDS_DataReaderResourceLimitsInstanceReplacementSettings::alive_instance_removal** (p. 1618) = **DDS_NO_INSTANCE_REMOVAL** (p. 1255)
- **DDS_DataReaderResourceLimitsInstanceReplacementSettings::disposed_instance_removal** (p. 1618) = **DDS_EMPTY_INSTANCE_REMOVAL** (p. 1255)
- **DDS_DataReaderResourceLimitsInstanceReplacementSettings::no_writers_instance_removal** (p. 1619) = **DDS_EMPTY_INSTANCE_REMOVAL** (p. 1255)

See also

DDS_DataReaderResourceLimitsQosPolicy::instance_replacement (p. 1635)

6.30.2 Field Documentation

6.30.2.1 alive_instance_removal

DDS_DataReaderInstanceRemovalKind **DDS_DataReaderResourceLimitsInstanceReplacementSettings::alive**↔
_instance_removal

Removal kind applied to alive (**DDS_ALIVE_INSTANCE_STATE** (p. 715)) instances.

[default] **DDS_NO_INSTANCE_REMOVAL** (p. 1255)

6.30.2.2 disposed_instance_removal

`DDS_DataReaderInstanceRemovalKind DDS_DataReaderResourceLimitsInstanceReplacementSettings::disposed←_instance_removal`

Removal kind applied to disposed (`DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE` (p. 715)) instances.

[default] `DDS_EMPTY_INSTANCE_REMOVAL` (p. 1255)

6.30.2.3 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. 715)) instances.

[default] `DDS_EMPTY_INSTANCE_REMOVAL` (p. 1255)

6.31 DDS_DataReaderResourceLimitsQosPolicy Struct Reference

<<*extension*>> (p. 795) Various settings that configure how a `DDS_DataReader` (p. 615) allocates and uses physical memory for internal resources.

Data Fields

- **DDS_Long max_remote_writers**
 <<*extension*>> (p. 795) The maximum number of remote writers from which a `DDS_DataReader` (p. 615) may read, including all instances.
- **DDS_Long max_remote_writers_per_instance**
 <<*extension*>> (p. 795) The maximum number of remote writers from which a `DDS_DataReader` (p. 615) may read a single instance.
- **DDS_Long max_samples_per_remote_writer**
 <<*extension*>> (p. 795) The maximum number of out-of-order samples from a given remote `DDS_DataWriter` (p. 485) that a `DDS_DataReader` (p. 615) may store when maintaining a reliable connection to the `DDS_DataWriter` (p. 485).
- **DDS_Long max_infos**
 <<*extension*>> (p. 795) The maximum number of info units that a `DDS_DataReader` (p. 615) can use to store `DDS←_SampleInfo` (p. 1985).
- **DDS_Long initial_remote_writers**
 <<*extension*>> (p. 795) The initial number of remote writers from which a `DDS_DataReader` (p. 615) may read, including all instances.
- **DDS_Long initial_remote_writers_per_instance**
 <<*extension*>> (p. 795) The initial number of remote writers from which a `DDS_DataReader` (p. 615) may read a single instance.
- **DDS_Long initial_infos**
 <<*extension*>> (p. 795) The initial number of info units that a `DDS_DataReader` (p. 615) can have, which are used to store `DDS_SampleInfo` (p. 1985).

- **DDS_Long initial_outstanding_reads**
 <<extension>> (p. 795) The initial number of outstanding calls to read/take (or one of their variants) on the same **DDS_DataReader** (p. 615) for which memory has not been returned by calling **FooDataReader_return_loan** (p. 646).
- **DDS_Long max_outstanding_reads**
 <<extension>> (p. 795) The maximum number of outstanding read/take calls (or one of their variants) on the same **DDS_DataReader** (p. 615) for which memory has not been returned by calling **FooDataReader_return_loan** (p. 646).
- **DDS_Long max_samples_per_read**
 <<extension>> (p. 795) The maximum number of data samples that the application can receive from the middleware in a single call to **FooDataReader_read** (p. 625) or **FooDataReader_take** (p. 626). If more data exists in the middleware, the application will need to issue multiple read/take calls.
- **DDS_Boolean disable_fragmentation_support**
 [DEPRECATED] (p. 965) <<extension>> (p. 795) Determines whether the **DDS_DataReader** (p. 615) can receive fragmented samples.
- **DDS_Long max_fragmented_samples**
 <<extension>> (p. 795) The maximum number of samples for which the **DDS_DataReader** (p. 615) may store fragments at a given point in time.
- **DDS_Long initial_fragmented_samples**
 <<extension>> (p. 795) The initial number of samples for which a **DDS_DataReader** (p. 615) may store fragments.
- **DDS_Long max_fragmented_samples_per_remote_writer**
 <<extension>> (p. 795) The maximum number of samples per remote writer for which a **DDS_DataReader** (p. 615) may store fragments.
- **DDS_Long max_fragments_per_sample**
 <<extension>> (p. 795) Maximum number of fragments for a single sample.
- **DDS_Boolean dynamically_allocate_fragmented_samples**
 <<extension>> (p. 795) Determines whether the **DDS_DataReader** (p. 615) pre-allocates storage for storing fragmented samples.
- **DDS_Long max_total_instances**
 <<extension>> (p. 795) Maximum number of instances for which a DataReader will keep state.
- **DDS_Long max_remote_virtual_writers**
 <<extension>> (p. 795) The maximum number of remote virtual writers from which a **DDS_DataReader** (p. 615) may read, including all instances.
- **DDS_Long initial_remote_virtual_writers**
 <<extension>> (p. 795) The initial number of remote virtual writers from which a **DDS_DataReader** (p. 615) may read, including all instances.
- **DDS_Long max_remote_virtual_writers_per_instance**
 <<extension>> (p. 795) The maximum number of virtual remote writers that can be associated with an instance.
- **DDS_Long initial_remote_virtual_writers_per_instance**
 <<extension>> (p. 795) The initial number of virtual remote writers per instance.
- **DDS_Long max_remote_writers_per_sample**
 <<extension>> (p. 795) The maximum number of remote writers allowed to write the same sample.
- **DDS_Long max_query_condition_filters**
 <<extension>> (p. 795) The maximum number of query condition filters a reader is allowed.
- **DDS_Long max_app_ack_response_length**
 <<extension>> (p. 795) Maximum length of application-level acknowledgment response data
- **DDS_Boolean keep_minimum_state_for_instances**
 <<extension>> (p. 795) Whether or not keep a minimum instance state for up to **DDS_DataReaderResourceLimits**↔**QosPolicy::max_total_instances** (p. 1629).
- **DDS_Long initial_topic_queries**
 <<extension>> (p. 795) The initial number of TopicQueries allocated by a **DDS_DataReader** (p. 615).

- **DDS_Long max_topic_queries**
 <<extension>> (p. 795) The maximum number of active *TopicQueries* that a **DDS_DataReader** (p. 615) can create.
- struct **DDS_AllocationSettings_t shmем_ref_transfer_mode_attached_segment_allocation**
 <<extension>> (p. 795) Allocation resource for the shared memory segments attached by the **DDS_DataReader** (p. 615).
- struct **DDS_DataReaderResourceLimitsInstanceReplacementSettings instance_replacement**
 <<extension>> (p. 795) Sets the kind of instances allowed to be replaced for each instance state (**DDS_InstanceStateKind** (p. 714)) when a *DataReader* reaches **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1951).
- struct **DDS_Duration_t autopurge_remote_not_alive_writer_delay**
 <<extension>> (p. 795) Maximum duration for which the **DDS_DataReader** (p. 615) will maintain information regarding a **DDS_DataWriter** (p. 485) once the **DDS_DataWriter** (p. 485) has become not alive.

6.31.1 Detailed Description

<<extension>> (p. 795) Various settings that configure how a **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615). By setting the initial size to the maximum size, you will prevent RTI Connext from dynamically allocating any memory after the creation of the **DDS_DataReader** (p. 615).

This QoS policy is an extension to the DDS standard.

Entity:

DDS_DataReader (p. 615)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

6.31.2 Field Documentation

6.31.2.1 max_remote_writers

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_remote_writers

<<**extension**>> (p. 795) The maximum number of remote writers from which a **DDS_DataReader** (p. 615) may read, including all instances.

[default] DDS_LENGTH_UNLIMITED (p. 1326)

[range] [1, 1 million] or DDS_LENGTH_UNLIMITED (p. 1326), >= 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. 1326).

Note: For efficiency, set max_remote_writers >= **DDS_DataReaderResourceLimitsQosPolicy::max_↵**
remote_writers_per_instance (p. 1622).

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.31.2.2 max_remote_writers_per_instance

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_remote_writers_per_instance

<<**extension**>> (p. 795) The maximum number of remote writers from which a **DDS_DataReader** (p. 615) may read a single instance.

[default] DDS_LENGTH_UNLIMITED (p. 1326)

[range] [1, 1024] or DDS_LENGTH_UNLIMITED (p. 1326), <= max_remote_writers or DDS_LENGTH_UNLIMITED (p. 1326), >= 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. 1326).

Note: For efficiency, set max_remote_writers_per_instance <= **DDS_DataReaderResourceLimitsQos_↵**
Policy::max_remote_writers (p. 1621)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.31.2.3 max_samples_per_remote_writer

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_samples_per_remote_writer

<<**extension**>> (p. 795) The maximum number of out-of-order samples from a given remote **DDS_DataWriter** (p. 485) that a **DDS_DataReader** (p. 615) may store when maintaining a reliable connection to the **DDS_DataWriter** (p. 485).

[default] **DDS_LENGTH_UNLIMITED** (p. 1326)

[range] [1, 100 million] or **DDS_LENGTH_UNLIMITED** (p. 1326), <= **DDS_ResourceLimitsQosPolicy::max_↵ samples** (p. 1951)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.31.2.4 max_infos

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_infos

<<**extension**>> (p. 795) The maximum number of info units that a **DDS_DataReader** (p. 615) can use to store **DDS_↵ _SampleInfo** (p. 1985).

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. 1326)

[range] [1, 1 million] or **DDS_LENGTH_UNLIMITED** (p. 1326), >= initial_infos

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.31.2.5 initial_remote_writers

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::initial_remote_writers

<<**extension**>> (p. 795) The initial number of remote writers from which a **DDS_DataReader** (p.615) 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. 1624).

Categories

Immutable (p.914), **Middleware Memory Management** (p.878)

6.31.2.6 initial_remote_writers_per_instance

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::initial_remote_writers_per_instance

<<**extension**>> (p. 795) The initial number of remote writers from which a **DDS_DataReader** (p.615) 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. 1623).

Categories

Immutable (p.914), **Middleware Memory Management** (p.878)

6.31.2.7 initial_infos

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::initial_infos

<<**extension**>> (p. 795) The initial number of info units that a **DDS_DataReader** (p. 615) can have, which are used to store **DDS_SampleInfo** (p. 1985).

[default] 32

[range] [1, 1 million], <= max_infos

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.31.2.8 initial_outstanding_reads

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::initial_outstanding_reads

<<**extension**>> (p. 795) The initial number of outstanding calls to read/take (or one of their variants) on the same **DDS_DataReader** (p. 615) for which memory has not been returned by calling **FooDataReader_return_loan** (p. 646).

[default] 2

[range] [1, 65536], <= max_outstanding_reads

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.31.2.9 max_outstanding_reads

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_outstanding_reads

<<**extension**>> (p. 795) The maximum number of outstanding read/take calls (or one of their variants) on the same **DDS_DataReader** (p. 615) for which memory has not been returned by calling **FooDataReader_return_loan** (p. 646).

[default] **DDS_LENGTH_UNLIMITED** (p. 1326)

[range] [1, 65536] or **DDS_LENGTH_UNLIMITED** (p. 1326), >= initial_outstanding_reads

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.31.2.10 max_samples_per_read

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_samples_per_read

<<*extension*>> (p. 795) The maximum number of data samples that the application can receive from the middleware in a single call to **FooDataReader_read** (p. 625) or **FooDataReader_take** (p. 626). 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. 1552) for more information about this feature.) When polling for data or using a **DDS_WaitSet** (p. 1371), 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. 914), **Middleware Memory Management** (p. 878)

6.31.2.11 disable_fragmentation_support

DDS_Boolean DDS_DataReaderResourceLimitsQosPolicy::disable_fragmentation_support

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Determines whether the **DDS_DataReader** (p. 615) can receive fragmented samples.

When fragmentation support is not needed, disabling fragmentation support will save some memory resources.

[default] **DDS_BOOLEAN_FALSE** (p. 1201)

Categories

Immutable (p. 914), **Fragmentation** (p. 866), **Sending Large Data** (p. 871)

6.31.2.12 max_fragmented_samples

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_fragmented_samples

<<**extension**>> (p. 795) The maximum number of samples for which the **DDS_DataReader** (p. 615) may store fragments at a given point in time.

At any given time, a **DDS_DataReader** (p. 615) 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 **DDS_DataWriter** (p. 485) 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. 1951).

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. 1626) is **DDS_↔ BOOLEAN_FALSE** (p. 1201).

[default] 1024

[range] [1, 1 million]

Categories

Immutable (p. 914), **Sample Memory Management** (p. 902), **Middleware Memory Management** (p. 878), **Fragmentation** (p. 866), **Sending Large Data** (p. 871)

6.31.2.13 initial_fragmented_samples

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::initial_fragmented_samples

<<**extension**>> (p. 795) The initial number of samples for which a **DDS_DataReader** (p. 615) may store fragments.

Only applies if **DDS_DataReaderResourceLimitsQosPolicy::disable_fragmentation_support** (p. 1626) is **DDS_↔ BOOLEAN_FALSE** (p. 1201).

[default] 4

[range] [1,1024], <= max_fragmented_samples

Categories

Immutable (p. 914), **Sample Memory Management** (p. 902), **Middleware Memory Management** (p. 878), **Fragmentation** (p. 866), **Sending Large Data** (p. 871)

6.31.2.14 max_fragmented_samples_per_remote_writer

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_fragmented_samples_per_remote_writer

<<**extension**>> (p. 795) The maximum number of samples per remote writer for which a **DDS_DataReader** (p. 615) may store fragments.

Logical limit so a single remote writer cannot consume all available resources.

Only applies if **DDS_DataReaderResourceLimitsQosPolicy::disable_fragmentation_support** (p. 1626) is **DDS_↔ BOOLEAN_FALSE** (p. 1201).

[default] 256

[range] [1, 1 million], <= max_fragmented_samples

Categories

Immutable (p. 914), **Sample Memory Management** (p. 902), **Middleware Memory Management** (p. 878), **Fragmentation** (p. 866), **Sending Large Data** (p. 871)

6.31.2.15 max_fragments_per_sample

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_fragments_per_sample

<<**extension**>> (p. 795) Maximum number of fragments for a single sample.

Only applies if **DDS_DataReaderResourceLimitsQosPolicy::disable_fragmentation_support** (p. 1626) is **DDS_↔ BOOLEAN_FALSE** (p. 1201).

[default] DDS_LENGTH_UNLIMITED (p. 1326)

[range] [1, 1 million] or **DDS_LENGTH_UNLIMITED** (p. 1326)

Categories

Immutable (p. 914), **Sample Memory Management** (p. 902), **Middleware Memory Management** (p. 878), **Fragmentation** (p. 866), **Sending Large Data** (p. 871)

6.31.2.16 dynamically_allocate_fragmented_samples

DDS_Boolean DDS_DataReaderResourceLimitsQosPolicy::dynamically_allocate_fragmented_samples

<<**extension**>> (p. 795) Determines whether the **DDS_DataReader** (p. 615) 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. 1201), the middleware will allocate memory upfront for storing fragments for up to **DDS_DataReaderResourceLimitsQosPolicy::initial_fragmented_samples** (p. 1627) samples. This memory may grow up to **DDS_DataReaderResourceLimitsQosPolicy::max_fragmented_samples** (p. 1626) if needed.

Only applies if **DDS_DataReaderResourceLimitsQosPolicy::disable_fragmentation_support** (p. 1626) is **DDS_BOOLEAN_FALSE** (p. 1201).

[default] **DDS_BOOLEAN_TRUE** (p. 1201)

Categories

Immutable (p. 914), **Sample Memory Management** (p. 902), **Middleware Memory Management** (p. 878), **Fragmentation** (p. 866), **Sending Large Data** (p. 871)

6.31.2.17 max_total_instances

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_total_instances

<<**extension**>> (p. 795) 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. 1951).

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. 626), **FooDataReader_read** (p. 625), and **FooDataReader_get_key_value** (p. 647).

The features Durable Reader State, MultiChannel DataWriters and RTI Persistence Service require RTI Connex 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 Connexx will check the resource limit `DDS_ResourceLimitsQosPolicy::max_instances` (p. 1951). If the limit is exceeded, RTI Connexx will drop the sample with the reason `LOST_BY_INSTANCES_LIMIT`. If the limit is not exceeded, RTI Connexx will check `max_total_instances`. If `max_total_instances` is exceeded, RTI Connexx 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. 1633) is false, and in that case should be left at the default value.

[default] `DDS_AUTO_MAX_TOTAL_INSTANCES` (p. 1256)

[range] [1, 1 million] or `DDS_LENGTH_UNLIMITED` (p. 1326) or `DDS_AUTO_MAX_TOTAL_INSTANCES` (p. 1256), $\geq$ `DDS_ResourceLimitsQosPolicy::max_instances` (p. 1951)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Instance Lifecycle** (p. 867)

6.31.2.18 `max_remote_virtual_writers`

DDS_Long `DDS_DataReaderResourceLimitsQosPolicy::max_remote_virtual_writers`

<<extension>> (p. 795) The maximum number of remote virtual writers from which a `DDS_DataReader` (p. 615) may read, including all instances.

When `DDS_PresentationQosPolicy::access_scope` (p. 1893) is set to `DDS_GROUP_PRESENTATION_QOS` (p. 1305), this value determines the maximum number of DataWriter groups that can be managed by the `DDS_Subscriber` (p. 572) containing this `DDS_DataReader` (p. 615).

Since the `DDS_Subscriber` (p. 572) may contain more than one `DDS_DataReader` (p. 615), only the setting of the first applies.

[default] `DDS_LENGTH_UNLIMITED` (p. 1326)

[range] [1, 1 million] or `DDS_LENGTH_UNLIMITED` (p. 1326), $\geq$ `initial_remote_virtual_writers`, $\geq$ `max_remote_virtual_writers_per_instance`

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.31.2.19 initial_remote_virtual_writers

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::initial_remote_virtual_writers

<<**extension**>> (p. 795) The initial number of remote virtual writers from which a **DDS_DataReader** (p. 615) may read, including all instances.

[default] 2

[range] [1, 1 million] or **DDS_LENGTH_UNLIMITED** (p. 1326), <= max_remote_virtual_writers

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.31.2.20 max_remote_virtual_writers_per_instance

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_remote_virtual_writers_per_instance

<<**extension**>> (p. 795) The maximum number of virtual remote writers that can be associated with an instance.

[default] **DDS_LENGTH_UNLIMITED** (p. 1326)

[range] [1, 1024] or **DDS_LENGTH_UNLIMITED** (p. 1326), >= 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 Connex 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 Connex 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. 914), **Middleware Memory Management** (p. 878)

6.31.2.21 initial_remote_virtual_writers_per_instance

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::initial_remote_virtual_writers_per_instance

<<**extension**>> (p. 795) 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. 914), **Middleware Memory Management** (p. 878)

6.31.2.22 max_remote_writers_per_sample

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_remote_writers_per_sample

<<**extension**>> (p. 795) 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. 914), **Middleware Memory Management** (p. 878)

6.31.2.23 max_query_condition_filters

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_query_condition_filters

<<**extension**>> (p. 795) 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_parameters` differ. Query conditions that differ only in their state masks will share the same query condition filter.

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Content Filtering** (p. 841)

6.31.2.24 max_app_ack_response_length

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_app_ack_response_length

<<**extension**>> (p. 795) 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. 914), **Middleware Memory Management** (p. 878), **Reliability** (p. 895), **Application Acknowledgement** (p. 835)

6.31.2.25 keep_minimum_state_for_instances

DDS_Boolean DDS_DataReaderResourceLimitsQosPolicy::keep_minimum_state_for_instances

<<*extension*>> (p. 795) Whether or not keep a minimum instance state for up to **DDS_DataReaderResourceLimitsQosPolicy::max_total_instances** (p. 1629).

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. 1629).

This additional state will only be kept for up to max_total_instances if this field is set to **DDS_BOOLEAN_TRUE** (p. 1201), 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. 1201)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Instance Lifecycle** (p. 867)

6.31.2.26 initial_topic_queries

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::initial_topic_queries

<<*extension*>> (p. 795) The initial number of TopicQueries allocated by a **DDS_DataReader** (p. 615).

[default] 1

See also

DDS_TopicQuery (p. 707)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Topic Queries** (p. 911)

6.31.2.27 max_topic_queries

DDS_Long DDS_DataReaderResourceLimitsQosPolicy::max_topic_queries

<<**extension**>> (p. 795) The maximum number of active TopicQueries that a **DDS_DataReader** (p. 615) can create.

Once this limit is reached, a **DDS_DataReader** (p. 615) can create more TopicQueries only if it deletes some of the previously created ones.

[default] **DDS_LENGTH_UNLIMITED** (p. 1326)

See also

DDS_TopicQuery (p. 707)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Topic Queries** (p. 911)

6.31.2.28 shm_ref_transfer_mode_attached_segment_allocation

struct DDS_AllocationSettings_t DDS_DataReaderResourceLimitsQosPolicy::shm_ref_transfer_mode_attached_segment_allocation

<<**extension**>> (p. 795) Allocation resource for the shared memory segments attached by the **DDS_DataReader** (p. 615).

The max_count does not limit the total number of shared memory segments used by the **DDS_DataReader** (p. 615). When this limit is hit, the **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615) will fail to attach to the new segment. This scenario will result in a sample loss.

[default] initial_count = **DDS_AUTO_COUNT** (p. 1284) (**DDS_DataReaderResourceLimitsQosPolicy::initial_remote_writers** (p. 1623)); max_count = **DDS_AUTO_COUNT** (p. 1284) (**DDS_DataReaderResourceLimitsQosPolicy::max_remote_writers** (p. 1621)); incremental_count = **DDS_AUTO_COUNT** (p. 1284) (0 if initial_count = max_count; -1 otherwise);

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 1537)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Sending Large Data** (p. 871)

6.31.2.29 instance_replacement

```
struct DDS_DataReaderResourceLimitsInstanceReplacementSettings DDS_DataReaderResourceLimitsQos↵
Policy::instance_replacement
```

<<**extension**>> (p. 795) Sets the kind of instances allowed to be replaced for each instance state (**DDS_Instance↵**
StateKind (p. 714)) when a DataReader reaches **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1951).

When **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1951) is reached, a **DDS_DataReader** (p. 615) 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. 714) separately. This means, for example, that a DataReader can choose to not allow replacing alive (**DDS_ALIVE_INSTANCE↵**
_STATE (p. 715)) instances but allow replacement of empty disposed (**DDS_NOT_ALIVE_DISPOSED_INSTANCE_↵**
STATE (p. 715)) 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 **DDS_DataReader** (p. 615) 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. 1302), 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. 618) and the instance will not be asserted into the DataReader queue.

[default]

- **DDS_DataReaderResourceLimitsInstanceReplacementSettings::alive_instance_removal** (p. 1618) = **DDS_NO_INSTANCE_REMOVAL** (p. 1255)
- **DDS_DataReaderResourceLimitsInstanceReplacementSettings::disposed_instance_removal** (p. 1618) = **DDS_EMPTY_INSTANCE_REMOVAL** (p. 1255)
- **DDS_DataReaderResourceLimitsInstanceReplacementSettings::no_writers_instance_removal** (p. 1619) = **DDS_EMPTY_INSTANCE_REMOVAL** (p. 1255)

See also

DDS_DataReaderResourceLimitsInstanceReplacementSettings (p. 1618)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Instance Lifecycle** (p. 867)

6.31.2.30 autopurge_remote_not_alive_writer_delay

```
struct DDS_Duration_t DDS_DataReaderResourceLimitsQosPolicy::autopurge_remote_not_alive_writer_↔
delay
```

<<**extension**>> (p. 795) Maximum duration for which the **DDS_DataReader** (p. 615) will maintain information regarding a **DDS_DataWriter** (p. 485) once the **DDS_DataWriter** (p. 485) has become not alive.

After this time elapses, the **DDS_DataReader** (p. 615) will purge all internal information regarding the not alive **DDS_↔DataWriter** (p. 485).

See also

DDS_LivelinessQosPolicy (p. 1823) for more information on when a **DDS_DataWriter** (p. 485) is considered not alive.

When set to **DDS_DURATION_AUTO** (p. 1209), this parameter is set to 10 times the value of **DDS_DiscoveryConfig_↔QosPolicy::participant_liveliness_lease_duration** (p. 1692).

This QoS only applies when the **DDS_DataReader** (p. 615) is using **DDS_RECOVER_INSTANCE_STATE_↔CONSISTENCY** (p. 1324) for the **DDS_ReliabilityQosPolicy::instance_state_consistency_kind** (p. 1940) QoS. When using **DDS_RECOVER_INSTANCE_STATE_CONSISTENCY** (p. 1324), a **DDS_DataReader** (p. 615) keeps state about all **DDS_DataWriter** (p. 485) 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 **DDS_DataWriter** (p. 485) is recovered. This can cause unbounded memory growth if that state is never purged and **DDS_DataWriter** (p. 485) 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 **DDS_DataWriter** (p. 485) and **DDS_DataReader** (p. 615) 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. 1621) QoS to a finite value. If that QoS is set to a finite value and the number of alive + not alive **DDS_DataWriter** (p. 485) entities reaches the limit when a new **DDS_DataWriter** (p. 485) is discovered, the oldest not alive **DDS_DataWriter** (p. 485) will be replaced.

[default] **DDS_DURATION_AUTO** (p. 1209)

[range] > 0 or **DDS_DURATION_AUTO** (p. 1209)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Instance Lifecycle** (p. 867), **Liveliness** (p. 874)

6.32 DDS_DataReaderSeq Struct Reference

Declares IDL sequence < **DDS_DataReader** (p. 615) > .

6.32.1 Detailed Description

Declares IDL sequence < **DDS_DataReader** (p. 615) > .

See also

FooSeq (p. 2113)

6.33 DDS_DataRepresentationIdSeq Struct Reference

Declares IDL sequence < **DDS_DataRepresentationId_t** (p. 1257) >

6.33.1 Detailed Description

Declares IDL sequence < **DDS_DataRepresentationId_t** (p. 1257) >

See also

DDS_DataRepresentationId_t (p. 1257)

6.34 DDS_DataRepresentationQosPolicy Struct Reference

<<*spec*>> (p. 790) This QoS policy contains a list of representation identifiers and compression settings used by **DDS_DataWriter** (p. 485) and **DDS_DataReader** (p. 615) entities to negotiate which data representation and compression settings to use.

Data Fields

- struct **DDS_DataRepresentationIdSeq** *value*
 <<*spec*>> (p. 790) *Sequence of representation identifiers.*
- struct **DDS_CompressionSettings_t** *compression_settings*
 <<*basic*>> (p. 783) <<*extension*>> (p. 795) *Structure that contains the compression settings.*

6.34.1 Detailed Description

<<*spec*>> (p. 790) This QoS policy contains a list of representation identifiers and compression settings used by **DDS_DataWriter** (p. 485) and **DDS_DataReader** (p. 615) entities to negotiate which data representation and compression settings to use.

Entity:

DDS_Topic (p. 175), **DDS_DataReader** (p. 615), **DDS_DataWriter** (p. 485)

Status:

DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 1229), **DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 1229)

Properties:

RxO (p. 1245) = YES
Changeable (p. 1245) = **UNTIL ENABLE** (p. 1245)

See also

DATA_REPRESENTATION (p. 1256)

This policy has request-offer semantics for both representation and compression. For representation, a **DDS_DataWriter** (p. 485) may only offer a single representation. Attempting to put multiple representations in a **DDS_DataWriter** (p. 485) will result in **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222). A **DDS_DataWriter** (p. 485) will use its offered policy to communicate with its matched **DDS_DataReader** (p. 615) entities. A **DDS_DataReader** (p. 615) requests one or more representations. If a **DDS_DataWriter** (p. 485) offers a representation that is contained within the sequence of the **DDS_DataReader** (p. 615), the offer satisfies the request and the policies are compatible. Otherwise, they are incompatible.

When representations are specified in the **DDS_TopicQos** (p. 2042), **DDS_Publisher_copy_from_topic_qos** (p. 462) copies the first element of the sequence, and **DDS_Subscriber_copy_from_topic_qos** (p. 591) copies the whole sequence.

For Compression, only the **DDS_CompressionSettings_t::compression_ids** (p. 1568) is propagated and a **DDS_DataWriter** (p. 485) may only offer a single compression id. Attempting to put multiple compression_ids in a **DDS_DataWriter** (p. 485) will result in **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222).

A **DDS_DataWriter** (p. 485) will use its offered compression algorithm to compress data that it sends to its matched **DDS_DataReader** (p. 615) entities. A **DDS_DataReader** (p. 615) requests zero or more compression algorithms. If a **DDS_DataWriter** (p. 485) offers a representation that is contained within the algorithms requested by the **DDS_DataReader** (p. 615), the offer satisfies the request and the policies are compatible. Otherwise, they are incompatible.

When compression IDs are specified in the **DDS_TopicQos** (p. 2042), **DDS_Publisher_copy_from_topic_qos** (p. 462) copies a single compression ID (see **DDS_CompressionSettings_t::compression_ids** (p. 1568)), and **DDS_Subscriber_copy_from_topic_qos** (p. 591) copies all of the compression_ids.

6.34.2 Field Documentation

6.34.2.1 value

```
struct DDS_DataRepresentationIdSeq DDS_DataRepresentationQosPolicy::value
```

<<*spec*>> (p. 790) Sequence of representation identifiers.

[default] For **DDS_Topic** (p. 175), **DDS_DataWriter** (p. 485), and **DDS_DataReader** (p. 615), a list with one element: **DDS_AUTO_DATA_REPRESENTATION** (p. 1258).

Note

An empty sequence is equivalent to a list with one element: **DDS_XCDR_DATA_REPRESENTATION** (p. 1257).

Categories

Immutable (p. 914), **Matching** (p. 963), **Discovery Metadata** (p. 958), **Type System** (p. 912)

6.34.2.2 compression_settings

```
struct DDS_CompressionSettings_t DDS_DataRepresentationQosPolicy::compression_settings
```

<<*basic*>> (p. 783) <<*extension*>> (p. 795) Structure that contains the compression settings.

compression_ids:

[default] The value depends on the entity

DDS_DataWriter (p. 485) and **DDS_Topic** (p. 175) are set to **DDS_COMPRESSION_ID_MASK_NONE** (p. 1260)

DDS_DataReader (p. 615) **DDS_COMPRESSION_ID_MASK_ALL** (p. 1260)

writer_compression_level:

[default] The value depends on the entity

DDS_DataWriter (p. 485) and **DDS_Topic** (p. 175) are set to **DDS_COMPRESSION_LEVEL_DEFAULT** (p. 1261)

DDS_DataReader (p. 615) NOT SUPPORTED

writer_compression_threshold:

[default] The value depends on the entity

DDS_DataWriter (p. 485) and **DDS_Topic** (p. 175) are set to **DDS_COMPRESSION_THRESHOLD_DEFAULT** (p. 1261)

DDS_DataReader (p. 615) NOT SUPPORTED

See also

DDS_CompressionSettings_t (p. 1568)

Categories

Immutable (p. 914), **Matching** (p. 963), **Discovery Metadata** (p. 958), **Sending Large Data** (p. 871)

6.35 DDS_DataTags Struct Reference

Definition of **DDS_DataTagQosPolicy** (p. 1264).

Data Fields

- struct **DDS_TagSeq** tags
<<*spec*>> (p. 790) *Sequence of data tags.*

6.35.1 Detailed Description

Definition of **DDS_DataTagQosPolicy** (p. 1264).

6.35.2 Field Documentation

6.35.2.1 tags

```
struct DDS_TagSeq DDS_DataTags::tags
```

<<*spec*>> (p. 790) *Sequence of data tags.*

[default] An empty list.

Categories

Immutable (p. 914), **Discovery Metadata** (p. 958), **Security** (p. 906)

6.36 DDS_DataWriterCacheStatus Struct Reference

<<*extension*>> (p. 1012) 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.

Data Fields

- **DDS_LongLong sample_count_peak**
The highest value of `DDS_DataWriterCacheStatus::sample_count` (p. 1642) 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. 715).
- **DDS_LongLong alive_instance_count_peak**
The highest value of `DDS_DataWriterCacheStatus::alive_instance_count` (p. 1643) 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. 715) (due to, for example, being disposed via the `FooDataWriter_dispose` (p. 502) operation).
- **DDS_LongLong disposed_instance_count_peak**
The highest value of `DDS_DataWriterCacheStatus::disposed_instance_count` (p. 1643) over the lifetime of the Data↔Writer.
- **DDS_LongLong unregistered_instance_count**
The number of instances currently in the DataWriter's queue that the DataWriter has unregistered from via the `FooData↔Writer_unregister_instance` (p. 493) operation.
- **DDS_LongLong unregistered_instance_count_peak**
The highest value of `DDS_DataWriterCacheStatus::unregistered_instance_count` (p. 1643) over the lifetime of the DataWriter.

6.36.1 Detailed Description

<<**extension**>> (p. 1012) 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:

DDS_DataWriter (p. 485)

6.36.2 Field Documentation

6.36.2.1 sample_count_peak

DDS_LongLong `DDS_DataWriterCacheStatus::sample_count_peak`

The highest value of `DDS_DataWriterCacheStatus::sample_count` (p. 1642) over the lifetime of the DataWriter.

6.36.2.2 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.

6.36.2.3 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. 715).

6.36.2.4 alive_instance_count_peak

DDS_LongLong DDS_DataWriterCacheStatus::alive_instance_count_peak

The highest value of **DDS_DataWriterCacheStatus::alive_instance_count** (p. 1643) over the lifetime of the Data↔Writer.

6.36.2.5 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. 715) (due to, for example, being disposed via the **FooDataWriter_dispose** (p. 502) operation).

6.36.2.6 disposed_instance_count_peak

DDS_LongLong DDS_DataWriterCacheStatus::disposed_instance_count_peak

The highest value of **DDS_DataWriterCacheStatus::disposed_instance_count** (p. 1643) over the lifetime of the DataWriter.

6.36.2.7 `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. 493) operation.

6.36.2.8 `unregistered_instance_count_peak`

DDS_LongLong `DDS_DataWriterCacheStatus::unregistered_instance_count_peak`

The highest value of **DDS_DataWriterCacheStatus::unregistered_instance_count** (p. 1643) over the lifetime of the DataWriter.

6.37 `DDS_DataWriterListener` Struct Reference

<<*interface*>> (p. 1013) **DDS_Listener** (p. 1816) for writer status.

Data Fields

- struct **DDS_Listener** as `listener`
*The superclass instance of this **DDS_DataWriterListener** (p. 1644).*
- **DDS_DataWriterListener_OfferedDeadlineMissedCallback** `on_offered_deadline_missed`
*Handles the **DDS_OFFERED_DEADLINE_MISSED_STATUS** (p. 1228) status.*
- **DDS_DataWriterListener_OfferedIncompatibleQosCallback** `on_offered_incompatible_qos`
*Handles the **DDS_OFFERED_INCOMPATIBLE_QOS_STATUS** (p. 1229) status.*
- **DDS_DataWriterListener_LivelinessLostCallback** `on_liveliness_lost`
*Handles the **DDS_LIVELINESS_LOST_STATUS** (p. 1231) status.*
- **DDS_DataWriterListener_PublicationMatchedCallback** `on_publication_matched`
*Handles the **DDS_PUBLICATION_MATCHED_STATUS** (p. 1232) status.*
- **DDS_DataWriterListener_ReliableWriterCacheChangedCallback** `on_reliable_writer_cache_changed`
 <<*extension*>> (p. 1012) *A change has occurred in the writer's cache of unacknowledged samples.*
- **DDS_DataWriterListener_ReliableReaderActivityChangedCallback** `on_reliable_reader_activity_changed`
 <<*extension*>> (p. 1012) *A matched reliable reader has become active or become inactive.*
- **DDS_DataWriterListener_SampleRemovedCallback** `on_sample_removed`
 <<*extension*>> (p. 1012) *Called when a sample is removed from the DataWriter queue.*
- **DDS_DataWriterListener_InstanceReplacedCallback** `on_instance_replaced`
 <<*extension*>> (p. 1012) *Notifies when an instance is replaced in DataWriter queue.*
- **DDS_DataWriterListener_OnApplicationAcknowledgmentCallback** `on_application_acknowledgment`
 <<*extension*>> (p. 1012) *Called when a sample is application-acknowledged*
- **DDS_DataWriterListener_ServiceRequestAcceptedCallback** `on_service_request_accepted`
 <<*extension*>> (p. 1012) *Called when a **DDS_ServiceRequest** (p. 2001) for the **DDS_TopicQuery** (p. 707) service is dispatched to this **DDS_DataWriter** (p. 485) for processing.*
- **DDS_DataWriterListener_OnEmptyVirtualWriterCallback** `on_empty_virtual_writer`
Callback function to indicate that a DataWriter does not have any samples left in the queue for a virtual writer.

6.37.1 Detailed Description

<<*interface*>> (p. 1013) **DDS_Listener** (p. 1816) for writer status.

Entity:

DDS_DataWriter (p. 485)

Status:

DDS_LIVELINESS_LOST_STATUS (p. 1231), **DDS_LivelinessLostStatus** (p. 1822);
DDS_OFFERED_DEADLINE_MISSED_STATUS (p. 1228), **DDS_OfferedDeadlineMissedStatus** (p. 1861);
DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 1229), **DDS_OfferedIncompatibleQosStatus** (p. 1862);
DDS_PUBLICATION_MATCHED_STATUS (p. 1232), **DDS_PublicationMatchedStatus** (p. 1914);
DDS_RELIABLE_READER_ACTIVITY_CHANGED_STATUS (p. 1234), **DDS_ReliableReaderActivity**↔
ChangedStatus (p. 1940);
DDS_RELIABLE_WRITER_CACHE_CHANGED_STATUS (p. 1234), **DDS_ReliableWriterCacheChanged**↔
Status (p. 1942);

See also

DDS_Listener (p. 1816)

Status Kinds (p. 1222)

Operations Allowed in Listener Callbacks (p. 1818)

6.37.2 Field Documentation

6.37.2.1 as_listener

```
struct DDS_Listener DDS_DataWriterListener::as_listener
```

The superclass instance of this **DDS_DataWriterListener** (p. 1644).

6.37.2.2 on_offered_deadline_missed

```
DDS_DataWriterListener_OfferedDeadlineMissedCallback DDS_DataWriterListener::on_offered_deadline↔  
_missed
```

Handles the **DDS_OFFERED_DEADLINE_MISSED_STATUS** (p. 1228) status.

This callback is called when the deadline that the **DDS_DataWriter** (p. 485) has committed through its **DEADLINE** (p. 1272) qos policy was not respected for a specific instance. This callback is called for each deadline period elapsed during which the **DDS_DataWriter** (p. 485) failed to provide data for an instance.

6.37.2.3 on_offered_incompatible_qos

DDS_DataWriterListener_OfferedIncompatibleQosCallback `DDS_DataWriterListener::on_offered_incompatible_qos`

Handles the **DDS_OFFERED_INCOMPATIBLE_QOS_STATUS** (p. 1229) status.

This callback is called when the **DDS_DataWriterQos** (p. 1666) of the **DDS_DataWriter** (p. 485) was incompatible with what was requested by a **DDS_DataReader** (p. 615). This callback is called when a **DDS_DataWriter** (p. 485) has discovered a **DDS_DataReader** (p. 615) for the same **DDS_Topic** (p. 175) and common partition, but with a requested QoS that is incompatible with that offered by the **DDS_DataWriter** (p. 485).

6.37.2.4 on_liveliness_lost

DDS_DataWriterListener_LivelinessLostCallback `DDS_DataWriterListener::on_liveliness_lost`

Handles the **DDS_LIVELINESS_LOST_STATUS** (p. 1231) status.

This callback is called when the liveliness that the **DDS_DataWriter** (p. 485) has committed through its **LIVELINESS** (p. 1296) qos policy was not respected; this **DDS_DataReader** (p. 615) entities will consider the **DDS_DataWriter** (p. 485) as no longer "alive/active". This callback will not be called when an already not alive **DDS_DataWriter** (p. 485) simply remains not alive for another liveliness period.

6.37.2.5 on_publication_matched

DDS_DataWriterListener_PublicationMatchedCallback `DDS_DataWriterListener::on_publication_matched`

Handles the **DDS_PUBLICATION_MATCHED_STATUS** (p. 1232) status.

This callback is called when the **DDS_DataWriter** (p. 485) has found a **DDS_DataReader** (p. 615) that matches the **DDS_Topic** (p. 175), has a common partition and compatible QoS, or has ceased to be matched with a **DDS_DataReader** (p. 615) that was previously considered to be matched.

6.37.2.6 on_reliable_writer_cache_changed

DDS_DataWriterListener_ReliableWriterCacheChangedCallback `DDS_DataWriterListener::on_reliable_writer_cache_changed`

<<*extension*>> (p. 1012) 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_ResourceLimitsQosPolicy::max_samples** (p. 1951)).
- The number of unacknowledged samples has reached **DDS_RtpsReliableWriterProtocol_t::high_watermark** (p. 1962) or **DDS_RtpsReliableWriterProtocol_t::low_watermark** (p. 1961).

6.37.2.7 on_reliable_reader_activity_changed

DDS_DataWriterListener_ReliableReaderActivityChangedCallback DDS_DataWriterListener::on_reliable_reader_activity_changed

<<**extension**>> (p. 1012) A matched reliable reader has become active or become inactive.

6.37.2.8 on_sample_removed

DDS_DataWriterListener_SampleRemovedCallback DDS_DataWriterListener::on_sample_removed

<<**extension**>> (p. 1012) 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. 1580) with **FooDataWriter_write_w_params** (p. 501), or if this writer uses **Zero Copy** (p. 210) transfer over shared memory" or **FlatData Topic-Types** (p. 209) "FlatData language binding".

See also

FooDataWriter_get_loan (p. 508)

6.37.2.9 on_instance_replaced

DDS_DataWriterListener_InstanceReplacedCallback DDS_DataWriterListener::on_instance_replaced

<<**extension**>> (p. 1012) Notifies when an instance is replaced in DataWriter queue.

This callback is called when an instance is replaced by the **DDS_DataWriter** (p. 485) 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. 505) 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. 505)
- **FooDataWriter_narrow** (p. 490)
- **FooDataWriter_as_datawriter** (p. 490)
- **FooDataWriter_create_data** (p. 506)
- **FooDataWriter_create_data_w_params** (p. 507)
- **FooDataWriter_delete_data** (p. 507)
- **FooDataWriter_delete_data_w_params** (p. 508)

- **DDS_DataWriter_as_entity** (p. 537)
- **DDS_DataWriter_get_matched_subscriptions** (p. 539)
- **DDS_DataWriter_is_matched_subscription_active** (p. 540)
- **DDS_DataWriter_get_matched_subscription_participant_data** (p. 542)
- **DDS_DataWriter_get_topic** (p. 542)
- **DDS_DataWriter_get_publisher** (p. 543)
- **DDS_DataWriter_is_sample_app_acknowledged** (p. 544)

6.37.2.10 on_application_acknowledgment

DDS_DataWriterListener_OnApplicationAcknowledgmentCallback `DDS_DataWriterListener::on_application_↔
_acknowledgment`

<<**extension**>> (p. 1012) Called when a sample is application-acknowledged

Applicable only when **DDS_ReliabilityQosPolicy::acknowledgment_kind** (p. 1939) = **DDS_APPLICATION_AUTO_↔
_ACKNOWLEDGMENT_MODE** (p. 1323) or **DDS_APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE** (p. 1324)

Called when a sample is application-level acknowledged. Provides identities of the sample and the acknowledging **DDS_DataReader** (p. 615). Also provides user-specified response data sent from the **DDS_DataReader** (p. 615) by the acknowledgment message.

6.37.2.11 on_service_request_accepted

DDS_DataWriterListener_ServiceRequestAcceptedCallback `DDS_DataWriterListener::on_service_request_↔
_accepted`

<<**extension**>> (p. 1012) Called when a **DDS_ServiceRequest** (p. 2001) for the **DDS_TopicQuery** (p. 707) service is dispatched to this **DDS_DataWriter** (p. 485) for processing.

See also

Topic Queries (p. 703)

6.37.2.12 on_empty_virtual_writer

DDS_DataWriterListener_OnEmptyVirtualWriterCallback `DDS_DataWriterListener::on_empty_virtual_↔
writer`

Callback function to indicate that a DataWriter does not have any samples left in the queue for a virtual writer.

6.38 DDS_DataWriterProtocolQosPolicy Struct Reference

<<**extension**>> (p. 795) Protocol that applies only to **DDS_DataWriter** (p. 485) instances.

Data Fields

- struct **DDS_GUID_t virtual_guid**
 <<**extension**>> (p. 795) *The virtual GUID (Global Unique Identifier).*
- **DDS_UnsignedLong rtps_object_id**
 <<**extension**>> (p. 795) *The RTPS Object ID.*
- **DDS_Boolean push_on_write**
 <<**extension**>> (p. 795) *Whether to push sample out when write is called.*
- **DDS_Boolean disable_positive_acks**
 <<**extension**>> (p. 795) *Controls whether or not the writer expects positive acknowledgements from matching readers.*
- **DDS_Boolean disable_inline_keyhash**
 <<**extension**>> (p. 795) *Controls whether or not a keyhash is propagated on the wire with each sample.*
- **DDS_Boolean serialize_key_with_dispose**
 <<**extension**>> (p. 795) *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. 795) *Controls whether or not a **DDS_DataWriter** (p. 485) receives **DDS_DataWriterListener**↔
 ::on_application_acknowledgment (p. 1648) notifications with an empty or invalid response.*
- struct **DDS_RtpsReliableWriterProtocol_t rtps_reliable_writer**
 <<**extension**>> (p. 795) *RTPS protocol-related configuration settings for the RTPS reliable writer associated with a **DDS_DataWriter** (p. 485). This parameter only has effect if both the writer and the matching reader are configured with **DDS_RELIABLE_RELIABILITY_QOS** (p. 1323) **DDS_ReliabilityQosPolicyKind** (p. 1322).*
- struct **DDS_SequenceNumber_t initial_virtual_sequence_number**
 <<**extension**>> (p. 795) *Determines, the initial virtual sequence number for this DataWriter.*

6.38.1 Detailed Description

<<**extension**>> (p. 795) Protocol that applies only to **DDS_DataWriter** (p. 485) 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. 1649) 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 **DDS_DataWriter** (p. 485) and **DDS_DataReader** (p. 615) 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. 1936) for more information on the per-Data↔Reader/DataWriter reliability configuration. **DDS_HistoryQosPolicy** (p. 1805) and **DDS_ResourceLimitsQosPolicy** (p. 1949) also play an important role in the DDS reliable protocol.

This QoS policy is an extension to the DDS standard.

Entity:

DDS_DataWriter (p. 485)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

6.38.2 Field Documentation

6.38.2.1 virtual_guid

```
struct DDS_GUID_t DDS_DataWriterProtocolQosPolicy::virtual_guid
```

<<**extension**>> (p. 795) The virtual GUID (Global Unique Identifier).

The virtual GUID is used to uniquely identify different incarnations of the same **DDS_DataWriter** (p. 485).

RTI Connext uses the virtual GUID to associate a persisted writer history to a specific **DDS_DataWriter** (p. 485).

The RTI Connext Persistence Service uses the virtual GUID to send samples on behalf of the original **DDS_DataWriter** (p. 485).

[default] **DDS_GUID_AUTO** (p. 1213)

Categories

Immutable (p. 914), **Discovery Metadata** (p. 958), **Data Availability** (p. 843)

6.38.2.2 rtps_object_id

```
DDS_UnsignedLong DDS_DataWriterProtocolQosPolicy::rtps_object_id
```

<<**extension**>> (p. 795) 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 recommendation is not to use automatic object ID assignment.

[default] **DDS_RTPS_AUTO_ID** (p. ??)

[range] [0,0x00ffffff]

Categories

Immutable (p. 914)

6.38.2.3 push_on_write

DDS_Boolean DDS_DataWriterProtocolQosPolicy::push_on_write

<<**extension**>> (p. 795) Whether to push sample out when write is called.

If set to **DDS_BOOLEAN_TRUE** (p. 1201) (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. 1201)

Categories

Immutable (p. 914), **Reliability** (p. 895), **Data Availability** (p. 843)

6.38.2.4 disable_positive_acks

DDS_Boolean DDS_DataWriterProtocolQosPolicy::disable_positive_acks

<<**extension**>> (p. 795) Controls whether or not the writer expects positive acknowledgements from matching readers.

If set to **DDS_BOOLEAN_TRUE** (p. 1201), the writer does not expect readers to 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. 1970), after which the sample is logically considered as positively acknowledged.

If set to **DDS_BOOLEAN_FALSE** (p. 1201) (the default), the writer expects to receive positive acknowledgements from its acknowledging readers (**DDS_DataReaderProtocolQosPolicy::disable_positive_acks** (p. 1598) = **DDS_BOOLEAN_FALSE** (p. 1201)) and it applies the keep-duration to its non-acknowledging readers (**DDS_DataReaderProtocolQosPolicy::disable_positive_acks** (p. 1598) = **DDS_BOOLEAN_TRUE** (p. 1201)).

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. 1201)

Categories

Immutable (p. 914), **Discovery Metadata** (p. 958), **Reliability** (p. 895)

6.38.2.5 disable_inline_keyhash

DDS_Boolean DDS_DataWriterProtocolQosPolicy::disable_inline_keyhash

<<**extension**>> (p. 795) 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 Connexx associates an internal 16-byte representation, called a keyhash.

When this field is **DDS_BOOLEAN_FALSE** (p. 1201), the keyhash is sent on the wire with every data instance.

When this field is **DDS_BOOLEAN_TRUE** (p. 1201), 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. 1201) is not compatible with using RTI Real-Time Connect or RTI Recorder.

[default] **DDS_BOOLEAN_FALSE** (p. 1201)

Categories

Immutable (p. 914), **Instance Lifecycle** (p. 867)

6.38.2.6 serialize_key_with_dispose

DDS_Boolean DDS_DataWriterProtocolQosPolicy::serialize_key_with_dispose

<<**extension**>> (p. 795) 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. 1201) if there are DataReaders where **DDS_DataReaderProtocolQosPolicy::propagate_dispose_of_unregistered_instances** (p. 1598) is also **DDS_BOOLEAN_TRUE** (p. 1201).

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. 647) 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. 1201), batching will not be compatible with RTI Connexx 4.3e, 4.4b, or 4.4c. The **DDS_DataReader** (p. 615) entities will receive incorrect data and/or encounter deserialization errors.

[default] **DDS_BOOLEAN_FALSE** (p. 1201)

Categories

Immutable (p. 914), **Instance Lifecycle** (p. 867)

6.38.2.7 propagate_app_ack_with_no_response

DDS_Boolean DDS_DataWriterProtocolQosPolicy::propagate_app_ack_with_no_response

<<**extension**>> (p. 795) Controls whether or not a **DDS_DataWriter** (p. 485) receives **DDS_DataWriterListener**↔::on_application_acknowledgment (p. 1648) notifications with an empty or invalid response.

When this field is set to **DDS_BOOLEAN_FALSE** (p. 1201), the callback **DDS_DataWriterListener**↔::on_application_acknowledgment (p. 1648) will not be invoked if the sample being acknowledged has an empty or invalid response.

[default] **DDS_BOOLEAN_TRUE** (p. 1201)

Categories

Immutable (p. 914), **Reliability** (p. 895), **Application Acknowledgement** (p. 835), **Durability** (p. 860)

6.38.2.8 rtps_reliable_writer

struct DDS_RtpsReliableWriterProtocol_t DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer

<<**extension**>> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with a **DDS_DataWriter** (p. 485). This parameter only has effect if both the writer and the matching reader are configured with **DDS_RELIABLE_RELIABILITY_QOS** (p. 1323) **DDS_ReliabilityQosPolicyKind** (p. 1322).

For details, refer to the **DDS_RtpsReliableWriterProtocol_t** (p. 1960)

[default] [default] See **DDS_RtpsReliableWriterProtocol_t** (p. 1960)

Categories

Immutable (p. 914), **Reliability** (p. 895)

6.38.2.9 initial_virtual_sequence_number

```
struct DDS_SequenceNumber_t DDS_DataWriterProtocolQosPolicy::initial_virtual_sequence_number
```

<<**extension**>> (p. 795) 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. 2102) in the **FooDataWriter_write_w_params** (p. 501).

[default] **DDS_AUTO_SEQUENCE_NUMBER** (p. 1218)

Categories

Immutable (p. 914)

6.39 DDS_DataWriterProtocolStatus Struct Reference

<<**extension**>> (p. 1012) The status of a writer's internal protocol related metrics, like the number of samples pushed, pulled, filtered; and status of wire protocol traffic.

Data Fields

- **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. 1657) 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. 1657) 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 DataWriter 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. 1658) 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. 1659) 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. 1659) 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. 1659) 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. 1660) 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. 1660) 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. 1661) 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. 1661) 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. 1661) 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. 1662) 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.

6.39.1 Detailed Description

<<**extension**>> (p. 1012) 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:

DDS_DataWriter (p. 485)

6.39.2 Field Documentation

6.39.2.1 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_DataWriterProtocolStatus::pushed_fragment_count** (p. 1664) statistic.

6.39.2.2 pushed_sample_count_change

DDS_LongLong DDS_DataWriterProtocolStatus::pushed_sample_count_change

The change in **DDS_DataWriterProtocolStatus::pushed_sample_count** (p. 1657) 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.

6.39.2.3 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.

6.39.2.4 pushed_sample_bytes_change

DDS_LongLong DDS_DataWriterProtocolStatus::pushed_sample_bytes_change

The change in **DDS_DataWriterProtocolStatus::pushed_sample_bytes** (p. 1657) 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.

6.39.2.5 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.

6.39.2.6 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.

6.39.2.7 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.

6.39.2.8 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.

6.39.2.9 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.

6.39.2.10 sent_heartbeat_count_change

DDS_LongLong DDS_DataWriterProtocolStatus::sent_heartbeat_count_change

The change in **DDS_DataWriterProtocolStatus::sent_heartbeat_count** (p. 1658) since the last time the status was read.

6.39.2.11 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.

6.39.2.12 sent_heartbeat_bytes_change

DDS_LongLong DDS_DataWriterProtocolStatus::sent_heartbeat_bytes_change

The change in **DDS_DataWriterProtocolStatus::sent_heartbeat_bytes** (p. 1659) since the last time the status was read.

6.39.2.13 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. 1650) is **DDS_BOOLEAN_FALSE** (p. 1201).

When data fragmentation is used, this statistic is incremented as fragments are written.

6.39.2.14 pulled_sample_count_change

DDS_LongLong DDS_DataWriterProtocolStatus::pulled_sample_count_change

The change in **DDS_DataWriterProtocolStatus::pulled_sample_count** (p. 1659) 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. 1650) is **DDS_BOOLEAN_FALSE** (p. 1201).

For large data, counts whole samples, not fragments.

6.39.2.15 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. 1650) is **DDS_BOOLEAN_FALSE** (p. 1201).

When data fragmentation is used, this statistic is incremented as fragments are written.

6.39.2.16 pulled_sample_bytes_change

DDS_LongLong DDS_DataWriterProtocolStatus::pulled_sample_bytes_change

The change in **DDS_DataWriterProtocolStatus::pulled_sample_bytes** (p. 1659) 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. 1650) is **DDS_BOOLEAN_FALSE** (p. 1201).

For large data, counts bytes of whole samples, not fragments.

6.39.2.17 received_ack_count

DDS_LongLong DDS_DataWriterProtocolStatus::received_ack_count

The number of ACKs from a remote DataReader received by a local DataWriter.

6.39.2.18 received_ack_count_change

DDS_LongLong DDS_DataWriterProtocolStatus::received_ack_count_change

The change in **DDS_DataWriterProtocolStatus::received_ack_count** (p. 1660) since the last time the status was read.

6.39.2.19 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.

6.39.2.20 received_ack_bytes_change

DDS_LongLong DDS_DataWriterProtocolStatus::received_ack_bytes_change

The change in **DDS_DataWriterProtocolStatus::received_ack_bytes** (p. 1660) since the last time the status was read.

6.39.2.21 received_nack_count

DDS_LongLong DDS_DataWriterProtocolStatus::received_nack_count

The number of NACKs from a remote DataReader received by a local DataWriter.

6.39.2.22 received_nack_count_change

DDS_LongLong DDS_DataWriterProtocolStatus::received_nack_count_change

The change in **DDS_DataWriterProtocolStatus::received_nack_count** (p. 1661) since the last time the status was read.

6.39.2.23 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.

6.39.2.24 received_nack_bytes_change

DDS_LongLong DDS_DataWriterProtocolStatus::received_nack_bytes_change

The change in **DDS_DataWriterProtocolStatus::received_nack_bytes** (p. 1661) since the last time the status was read.

6.39.2.25 sent_gap_count

DDS_LongLong DDS_DataWriterProtocolStatus::sent_gap_count

The number of GAPS sent from local DataWriter to matching remote DataReaders.

6.39.2.26 sent_gap_count_change

DDS_LongLong DDS_DataWriterProtocolStatus::sent_gap_count_change

The change in **DDS_DataWriterProtocolStatus::sent_gap_count** (p. 1661) since the last time the status was read.

6.39.2.27 sent_gap_bytes

DDS_LongLong DDS_DataWriterProtocolStatus::sent_gap_bytes

The number of bytes of GAPS sent from local DataWriter to matching remote DataReaders.

6.39.2.28 sent_gap_bytes_change

DDS_LongLong DDS_DataWriterProtocolStatus::sent_gap_bytes_change

The change in **DDS_DataWriterProtocolStatus::sent_gap_bytes** (p. 1662) since the last time the status was read.

6.39.2.29 rejected_sample_count

DDS_LongLong DDS_DataWriterProtocolStatus::rejected_sample_count

[Not supported.]

6.39.2.30 rejected_sample_count_change

DDS_LongLong DDS_DataWriterProtocolStatus::rejected_sample_count_change

[Not supported.]

6.39.2.31 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. 1973) to `DDS_RtpsReliableWriterProtocol_t::max_send_window_size` (p. 1973).

6.39.2.32 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.

6.39.2.33 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.

6.39.2.34 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.

6.39.2.35 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.

6.39.2.36 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.

6.39.2.37 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.

6.39.2.38 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.

6.39.2.39 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. 1651) is set.

Sequence number of the first sample kept in the DataWriter's queue whose keep_duration (applied when **DDS_Data↵WriterProtocolQosPolicy::disable_positive_acks** (p. 1651) is set) has not yet elapsed.

Applies only for local DataWriter status.

6.39.2.40 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.

6.39.2.41 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.

6.39.2.42 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.

6.39.2.43 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.

6.39.2.44 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.

6.39.2.45 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.

6.40 DDS_DataWriterQos Struct Reference

QoS policies supported by a **DDS_DataWriter** (p. 485) entity.

Data Fields

- struct **DDS_DurabilityQosPolicy** `durability`
*Durability policy, **DURABILITY** (p. 1285).*
- struct **DDS_DurabilityServiceQosPolicy** `durability_service`
*DurabilityService policy, **DURABILITY_SERVICE** (p. 1289).*
- struct **DDS_DeadlineQosPolicy** `deadline`
*<<basic>> (p. 783) Deadline policy, **DEADLINE** (p. 1272).*
- struct **DDS_LatencyBudgetQosPolicy** `latency_budget`
*Latency budget policy, **LATENCY_BUDGET** (p. 1294).*
- struct **DDS_LivelinessQosPolicy** `liveliness`
*Liveliness policy, **LIVELINESS** (p. 1296).*
- struct **DDS_ReliabilityQosPolicy** `reliability`
*Reliability policy, **RELIABILITY** (p. 1321).*
- struct **DDS_DestinationOrderQosPolicy** `destination_order`
*Destination order policy, **DESTINATION_ORDER** (p. 1273).*
- struct **DDS_HistoryQosPolicy** `history`
*<<basic>> (p. 783) History policy, **HISTORY** (p. 1292).*
- struct **DDS_ResourceLimitsQosPolicy** `resource_limits`
*Resource limits policy, **RESOURCE_LIMITS** (p. 1325).*
- struct **DDS_TransportPriorityQosPolicy** `transport_priority`
*Transport priority policy, **TRANSPORT_PRIORITY** (p. 1337).*
- struct **DDS_LifespanQosPolicy** `lifespan`
*<<basic>> (p. 783) Lifespan policy, **LIFESPAN** (p. 1295).*
- struct **DDS_UserDataQosPolicy** `user_data`
*User data policy, **USER_DATA** (p. 1343).*
- struct **DDS_OwnershipQosPolicy** `ownership`
*<<basic>> (p. 783) Ownership policy, **OWNERSHIP** (p. 1301).*
- struct **DDS_OwnershipStrengthQosPolicy** `ownership_strength`
*<<basic>> (p. 783) Ownership strength policy, **OWNERSHIP_STRENGTH** (p. 1302).*
- struct **DDS_WriterDataLifecycleQosPolicy** `writer_data_lifecycle`
*Writer data lifecycle policy, **WRITER_DATA_LIFECYCLE** (p. 1344).*
- struct **DDS_DataRepresentationQosPolicy** `representation`
*Data representation policy, **DATA_REPRESENTATION** (p. 1256).*
- **DDS_DataTagQosPolicy** `data_tags`
*DataTag policy, **DATA_TAG** (p. 1263).*
- struct **DDS_DataWriterResourceLimitsQosPolicy** `writer_resource_limits`
*<<extension>> (p. 1012) Writer resource limits policy, **DATA_WRITER_RESOURCE_LIMITS** (p. 1269).*
- struct **DDS_DataWriterProtocolQosPolicy** `protocol`
*<<extension>> (p. 1012) **DDS_DataWriter** (p. 485) protocol policy, **DATA_WRITER_PROTOCOL** (p. 1268).*
- struct **DDS_TransportSelectionQosPolicy** `transport_selection`
*<<extension>> (p. 1012) Transport plugin selection policy, **TRANSPORT_SELECTION** (p. 1338).*

- struct **DDS_TransportUnicastQosPolicy** **unicast**
 <<extension>> (p. 1012) Unicast transport policy, **TRANSPORT_UNICAST** (p. 1339).
- struct **DDS_PublishModeQosPolicy** **publish_mode**
 <<extension>> (p. 1012) Publish mode policy, **PUBLISH_MODE** (p. 1317).
- struct **DDS_PropertyQosPolicy** **property**
 <<basic>> (p. 783) <<extension>> (p. 1012) Property policy, **PROPERTY** (p. 1306). See also *Property Reference Guide*.
- struct **DDS_ServiceQosPolicy** **service**
 <<extension>> (p. 1012) Service policy, **SERVICE** (p. 1326).
- struct **DDS_BatchQosPolicy** **batch**
 <<extension>> (p. 1012) Batch policy, **BATCH** (p. 1251).
- struct **DDS_MultiChannelQosPolicy** **multi_channel**
 <<extension>> (p. 1012) Multi channel policy, **MULTICHANNEL** (p. 1300).
- struct **DDS_AvailabilityQosPolicy** **availability**
 <<extension>> (p. 1012) Availability policy, **AVAILABILITY** (p. 1250).
- struct **DDS_EntityNameQosPolicy** **publication_name**
 <<extension>> (p. 1012) EntityName policy, **ENTITY_NAME** (p. 1290).
- struct **DDS_TopicQueryDispatchQosPolicy** **topic_query_dispatch**
 <<extension>> (p. 1012) Topic Query dispatch policy, **TOPIC_QUERY_DISPATCH** (p. 1330).
- struct **DDS_DataWriterTransferModeQosPolicy** **transfer_mode**
 <<extension>> (p. 1012) TransferMode policy, **DATA_WRITER_TRANSFER_MODE** (p. 1272).
- struct **DDS_TypeSupportQosPolicy** **type_support**
 <<extension>> (p. 1012) Type support data, **TYPESUPPORT** (p. 1341).

6.40.1 Detailed Description

QoS policies supported by a **DDS_DataWriter** (p. 485) entity.

You must set certain members in a consistent manner:

- **DDS_DataWriterQos::history** (p. 1669) .depth <= **DDS_DataWriterQos::resource_limits** (p. 1669) .max_↵ samples_per_instance
- **DDS_DataWriterQos::resource_limits** (p. 1669) .max_samples_per_instance <= **DDS_DataWriterQos**↵
::resource_limits (p. 1669) .max_samples
- **DDS_DataWriterQos::resource_limits** (p. 1669) .initial_samples <= **DDS_DataWriterQos::resource_limits**
 (p. 1669) .max_samples
- **DDS_DataWriterQos::resource_limits** (p. 1669) .initial_instances <= **DDS_DataWriterQos::resource_limits**
 (p. 1669) .max_instances
- length of **DDS_DataWriterQos::user_data** (p. 1670) .value <= **DDS_DomainParticipantQos::resource_limits**
 (p. 1729) .writer_user_data_max_length

If any of the above are not true, **DDS_DataWriter_set_qos** (p. 550) and **DDS_DataWriter_set_qos_with_profile** (p. 551) and **DDS_Publisher_set_default_datawriter_qos** (p. 450) and **DDS_Publisher_set_default_datawriter**↵
qos_with_profile (p. 450) will fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222) and **DDS_Publisher**↵
create_datawriter (p. 453) and **DDS_Publisher_create_datawriter_with_profile** (p. 455) and will return NULL.

Entity:

DDS_DataWriter (p. 485)

See also

All QoS Policies (p. 1238) allowed ranges within each Qos.

6.40.2 Field Documentation

6.40.2.1 durability

```
struct DDS_DurabilityQosPolicy DDS_DataWriterQos::durability
```

Durability policy, **DURABILITY** (p. 1285).

6.40.2.2 durability_service

```
struct DDS_DurabilityServiceQosPolicy DDS_DataWriterQos::durability_service
```

DurabilityService policy, **DURABILITY_SERVICE** (p. 1289).

6.40.2.3 deadline

```
struct DDS_DeadlineQosPolicy DDS_DataWriterQos::deadline
```

<<*basic*>> (p. 783) Deadline policy, **DEADLINE** (p. 1272).

6.40.2.4 latency_budget

```
struct DDS_LatencyBudgetQosPolicy DDS_DataWriterQos::latency_budget
```

Latency budget policy, **LATENCY_BUDGET** (p. 1294).

6.40.2.5 liveliness

```
struct DDS_LivelinessQosPolicy DDS_DataWriterQos::liveliness
```

Liveliness policy, **LIVELINESS** (p. 1296).

6.40.2.6 reliability

```
struct DDS_ReliabilityQosPolicy DDS_DataWriterQos::reliability
```

Reliability policy, **RELIABILITY** (p. 1321).

6.40.2.7 destination_order

```
struct DDS_DestinationOrderQosPolicy DDS_DataWriterQos::destination_order
```

Destination order policy, **DESTINATION_ORDER** (p. 1273).

6.40.2.8 history

```
struct DDS_HistoryQosPolicy DDS_DataWriterQos::history
```

<<**basic**>> (p. 783) History policy, **HISTORY** (p. 1292).

6.40.2.9 resource_limits

```
struct DDS_ResourceLimitsQosPolicy DDS_DataWriterQos::resource_limits
```

Resource limits policy, **RESOURCE_LIMITS** (p. 1325).

6.40.2.10 transport_priority

```
struct DDS_TransportPriorityQosPolicy DDS_DataWriterQos::transport_priority
```

Transport priority policy, **TRANSPORT_PRIORITY** (p. 1337).

6.40.2.11 lifespan

```
struct DDS_LifespanQosPolicy DDS_DataWriterQos::lifespan
```

<<**basic**>> (p. 783) Lifespan policy, **LIFESPAN** (p. 1295).

6.40.2.12 user_data

```
struct DDS_UserDataQosPolicy DDS_DataWriterQos::user_data
```

User data policy, **USER_DATA** (p. 1343).

6.40.2.13 ownership

```
struct DDS_OwnershipQosPolicy DDS_DataWriterQos::ownership
```

<<**basic**>> (p. 783) Ownership policy, **OWNERSHIP** (p. 1301).

6.40.2.14 ownership_strength

```
struct DDS_OwnershipStrengthQosPolicy DDS_DataWriterQos::ownership_strength
```

<<**basic**>> (p. 783) Ownership strength policy, **OWNERSHIP_STRENGTH** (p. 1302).

6.40.2.15 writer_data_lifecycle

```
struct DDS_WriterDataLifecycleQosPolicy DDS_DataWriterQos::writer_data_lifecycle
```

Writer data lifecycle policy, **WRITER_DATA_LIFECYCLE** (p. 1344).

6.40.2.16 representation

```
struct DDS_DataRepresentationQosPolicy DDS_DataWriterQos::representation
```

Data representation policy, **DATA_REPRESENTATION** (p. 1256).

6.40.2.17 data_tags

`DDS_DataTagQosPolicy DDS_DataWriterQos::data_tags`

DataTag policy, **DATA_TAG** (p. 1263).

6.40.2.18 writer_resource_limits

`struct DDS_DataWriterResourceLimitsQosPolicy DDS_DataWriterQos::writer_resource_limits`

<<*extension*>> (p. 1012) Writer resource limits policy, **DATA_WRITER_RESOURCE_LIMITS** (p. 1269).

6.40.2.19 protocol

`struct DDS_DataWriterProtocolQosPolicy DDS_DataWriterQos::protocol`

<<*extension*>> (p. 1012) **DDS_DataWriter** (p. 485) protocol policy, **DATA_WRITER_PROTOCOL** (p. 1268)

6.40.2.20 transport_selection

`struct DDS_TransportSelectionQosPolicy DDS_DataWriterQos::transport_selection`

<<*extension*>> (p. 1012) Transport plugin selection policy, **TRANSPORT_SELECTION** (p. 1338).

Specifies the transports available for use by the **DDS_DataWriter** (p. 485).

6.40.2.21 unicast

`struct DDS_TransportUnicastQosPolicy DDS_DataWriterQos::unicast`

<<*extension*>> (p. 1012) Unicast transport policy, **TRANSPORT_UNICAST** (p. 1339).

Specifies the unicast transport interfaces and ports on which **messages** can be received.

The unicast interfaces are used to receive messages from **DDS_DataReader** (p. 615) entities in the domain.

6.40.2.22 publish_mode

```
struct DDS_PublishModeQosPolicy DDS_DataWriterQos::publish_mode
```

<<**extension**>> (p. 1012) Publish mode policy, **PUBLISH_MODE** (p. 1317).

Determines whether the **DDS_DataWriter** (p. 485) publishes data synchronously or asynchronously and how.

6.40.2.23 property

```
struct DDS_PropertyQosPolicy DDS_DataWriterQos::property
```

<<**basic**>> (p. 783) <<**extension**>> (p. 1012) Property policy, **PROPERTY** (p. 1306). See also `Property Reference Guide`.

6.40.2.24 service

```
struct DDS_ServiceQosPolicy DDS_DataWriterQos::service
```

<<**extension**>> (p. 1012) Service policy, **SERVICE** (p. 1326).

6.40.2.25 batch

```
struct DDS_BatchQosPolicy DDS_DataWriterQos::batch
```

<<**extension**>> (p. 1012) Batch policy, **BATCH** (p. 1251).

6.40.2.26 multi_channel

```
struct DDS_MultiChannelQosPolicy DDS_DataWriterQos::multi_channel
```

<<**extension**>> (p. 1012) Multi channel policy, **MULTICHANNEL** (p. 1300).

6.40.2.27 availability

```
struct DDS_AvailabilityQosPolicy DDS_DataWriterQos::availability
```

<<**extension**>> (p. 1012) Availability policy, **AVAILABILITY** (p. 1250).

6.40.2.28 publication_name

```
struct DDS_EntityNameQosPolicy DDS_DataWriterQos::publication_name
```

<<*extension*>> (p. 1012) EntityName policy, **ENTITY_NAME** (p. 1290).

6.40.2.29 topic_query_dispatch

```
struct DDS_TopicQueryDispatchQosPolicy DDS_DataWriterQos::topic_query_dispatch
```

<<*extension*>> (p. 1012) Topic Query dispatch policy, **TOPIC_QUERY_DISPATCH** (p. 1330).

6.40.2.30 transfer_mode

```
struct DDS_DataWriterTransferModeQosPolicy DDS_DataWriterQos::transfer_mode
```

<<*extension*>> (p. 1012) TransferMode policy, **DATA_WRITER_TRANSFER_MODE** (p. 1272).

6.40.2.31 type_support

```
struct DDS_TypeSupportQosPolicy DDS_DataWriterQos::type_support
```

<<*extension*>> (p. 1012) Type support data, **TYPESUPPORT** (p. 1341).

Optional value that is passed to a type plugin's on_endpoint_attached and serialization functions.

6.41 DDS_DataWriterResourceLimitsQosPolicy Struct Reference

<<*extension*>> (p. 795) Various settings that configure how a **DDS_DataWriter** (p. 485) allocates and uses physical memory for internal resources.

Data Fields

- **DDS_Long initial_concurrent_blocking_threads**
*[DEPRECATED] (p. 965) <<extension>> (p. 795) The initial number of threads that are allowed to concurrently block on write call on the same **DDS_DataWriter** (p. 485).*
- **DDS_Long max_concurrent_blocking_threads**
*[DEPRECATED] (p. 965) <<extension>> (p. 795) The maximum number of threads that are allowed to concurrently block on write call on the same **DDS_DataWriter** (p. 485).*
- **DDS_Long max_remote_reader_filters**
<<extension>> (p. 795) The maximum number of remote DataReaders for which the **DDS_DataWriter** (p. 485) will perform content-based filtering.
- **DDS_Long initial_batches**
<<extension>> (p. 795) Represents the initial number of batches a **DDS_DataWriter** (p. 485) will manage.
- **DDS_Long max_batches**
<<extension>> (p. 795) Represents the maximum number of batches a **DDS_DataWriter** (p. 485) will manage.
- **DDS_DataWriterResourceLimitsInstanceReplacementKind instance_replacement**
<<extension>> (p. 795) Sets the kinds of instances allowed to be replaced when instance resource limits are reached.
- **DDS_Boolean replace_empty_instances**
<<extension>> (p. 795) Whether or not to replace empty instances during instance replacement
- **DDS_Boolean autoregister_instances**
[DEPRECATED] (p. 965) <<extension>> (p. 795) Whether or not to automatically register new instances
- **DDS_Long initial_virtual_writers**
<<extension>> (p. 795) The initial number of virtual writers supported by a **DDS_DataWriter** (p. 485)
- **DDS_Long max_virtual_writers**
<<extension>> (p. 795) The maximum number of virtual writers supported by a **DDS_DataWriter** (p. 485)
- **DDS_Long max_remote_readers**
<<extension>> (p. 795) The maximum number of remote readers supported by a **DDS_DataWriter** (p. 485)
- **DDS_Long max_app_ack_remote_readers**
<<extension>> (p. 795) The maximum number of application-level acknowledging remote readers supported by a **DDS_DataWriter** (p. 485)
- **DDS_Long initial_active_topic_queries**
<<extension>> (p. 795) Represents the initial number of active topic queries a **DDS_DataWriter** (p. 485) will manage.
- **DDS_Long max_active_topic_queries**
<<extension>> (p. 795) Represents the maximum number of active topic queries a **DDS_DataWriter** (p. 485) will manage.
- struct **DDS_AllocationSettings_t writer_loaned_sample_allocation**
<<extension>> (p. 795) Represents the allocation settings of loaned samples managed by a **DDS_DataWriter** (p. 485).
- **DDS_Boolean initialize_writer_loaned_sample**
<<extension>> (p. 795) Whether or not to initialize loaned samples returned by a **DDS_DataWriter** (p. 485)

6.41.1 Detailed Description

<<extension>> (p. 795) Various settings that configure how a **DDS_DataWriter** (p. 485) 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. 496) on the same **DDS_DataWriter** (p. 485), 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 **DDS_DataWriter** (p. 485). By setting the initial size to the maximum size, you will prevent RTI Connex from dynamically allocating any memory after the creation of the **DDS_DataWriter** (p. 485).

This QoS policy is an extension to the DDS standard.

Entity:

DDS_DataWriter (p. 485)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

6.41.2 Field Documentation

6.41.2.1 initial_concurrent_blocking_threads

DDS_Long DDS_DataWriterResourceLimitsQosPolicy::initial_concurrent_blocking_threads

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) The initial number of threads that are allowed to concurrently block on write call on the same **DDS_DataWriter** (p. 485).

This value only applies if **DDS_HistoryQosPolicy** (p. 1805) has its kind set to **DDS_KEEP_ALL_HISTORY_QOS** (p. 1293) and **DDS_ReliabilityQosPolicy::max_blocking_time** (p. 1939) is > 0.

[default] 1

[range] [1, 10000], <= max_concurrent_blocking_threads

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Concurrency** (p. 839)

6.41.2.2 max_concurrent_blocking_threads

DDS_Long DDS_DataWriterResourceLimitsQosPolicy::max_concurrent_blocking_threads

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) The maximum number of threads that are allowed to concurrently block on write call on the same **DDS_DataWriter** (p. 485).

This value only applies if **DDS_HistoryQosPolicy** (p. 1805) has its kind set to **DDS_KEEP_ALL_HISTORY_QOS** (p. 1293) and **DDS_ReliabilityQosPolicy::max_blocking_time** (p. 1939) is > 0.

[default] **DDS_LENGTH_UNLIMITED** (p. 1326)

[range] [1, 10000] or **DDS_LENGTH_UNLIMITED** (p. 1326), >= initial_concurrent_blocking_threads

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Concurrency** (p. 839)

6.41.2.3 max_remote_reader_filters

DDS_Long DDS_DataWriterResourceLimitsQosPolicy::max_remote_reader_filters

<<*extension*>> (p. 795) The maximum number of remote DataReaders for which the **DDS_DataWriter** (p. 485) will perform content-based filtering.

[default] **DDS_LENGTH_UNLIMITED** (p. 1326)

[range] [0, (2³¹)-2] or **DDS_LENGTH_UNLIMITED** (p. 1326).

0: The **DDS_DataWriter** (p. 485) will not perform filtering for any **DDS_DataReader** (p. 615).

1 to (2³¹)-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. 1326): The DataWriter will filter for up to (2³¹)-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. 914), **Middleware Memory Management** (p. 878), **Content Filtering** (p. 841)

6.41.2.4 initial_batches

`DDS_Long DDS_DataWriterResourceLimitsQosPolicy::initial_batches`

<<*extension*>> (p. 795) Represents the initial number of batches a **DDS_DataWriter** (p. 485) will manage.

[default] 8

[range] [1,100 million]

See also

DDS_BatchQosPolicy (p. 1552)

Categories

Immutable (p. 914), **Sample Memory Management** (p. 902), **Middleware Memory Management** (p. 878), **Batching** (p. 838)

6.41.2.5 max_batches

`DDS_Long DDS_DataWriterResourceLimitsQosPolicy::max_batches`

<<*extension*>> (p. 795) Represents the maximum number of batches a **DDS_DataWriter** (p. 485) will manage.

[default] **DDS_LENGTH_UNLIMITED** (p. 1326)

When batching is enabled, the maximum number of samples that a **DDS_DataWriter** (p. 485) can store is limited by this value and **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1951).

[range] [1,100 million] or **DDS_LENGTH_UNLIMITED** (p. 1326) $\geq$ **DDS_RtpsReliableWriterProtocol_t** $\leftrightarrow$ **::heartbeats_per_max_samples** (p. 1967) if batching is enabled

See also

DDS_BatchQosPolicy (p. 1552)

Categories

Immutable (p. 914), **Sample Memory Management** (p. 902), **Middleware Memory Management** (p. 878), **Batching** (p. 838)

6.41.2.6 instance_replacement

DDS_DataWriterResourceLimitsInstanceReplacementKind DDS_DataWriterResourceLimitsQosPolicy::instance↵↵
_replacement

<<**extension**>> (p. 795) Sets the kinds of instances allowed to be replaced when instance resource limits are reached.

When a **DDS_DataWriter** (p. 485)'s number of active instances is greater than **DDS_ResourceLimitsQosPolicy::max↵↵::max_instances** (p. 1951), 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. 1270)

See also

DDS_DataWriterResourceLimitsInstanceReplacementKind (p. 1269)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Instance Lifecycle** (p. 867)

6.41.2.7 replace_empty_instances

DDS_Boolean DDS_DataWriterResourceLimitsQosPolicy::replace_empty_instances

<<**extension**>> (p. 795) Whether or not to replace empty instances during instance replacement

When a **DDS_DataWriter** (p. 485) has more active instances than allowed by **DDS_ResourceLimitsQosPolicy::max↵↵_instances** (p. 1951), 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. 1201), then a **DDS_↵↵DataWriter** (p. 485) will first try reclaiming empty instances, before trying to replace whatever is specified by **DDS_↵↵DataWriterResourceLimitsQosPolicy::instance_replacement** (p. 1677).

[default] **DDS_BOOLEAN_FALSE** (p. 1201)

See also

DDS_DataWriterResourceLimitsInstanceReplacementKind (p. 1269)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Instance Lifecycle** (p. 867)

6.41.2.8 autoregister_instances

`DDS_Boolean DDS_DataWriterResourceLimitsQosPolicy::autoregister_instances`

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Whether or not to automatically register new instances

[default] `DDS_BOOLEAN_FALSE` (p. 1201)

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. 496)

Categories

Immutable (p. 914), **Instance Lifecycle** (p. 867)

6.41.2.9 initial_virtual_writers

`DDS_Long DDS_DataWriterResourceLimitsQosPolicy::initial_virtual_writers`

<<*extension*>> (p. 795) The initial number of virtual writers supported by a **DDS_DataWriter** (p. 485)

[default] 1

[range] [1, 1000000], or **DDS_LENGTH_UNLIMITED** (p. 1326)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.41.2.10 max_virtual_writers

`DDS_Long DDS_DataWriterResourceLimitsQosPolicy::max_virtual_writers`

<<*extension*>> (p. 795) The maximum number of virtual writers supported by a **DDS_DataWriter** (p. 485)

Sets the maximum number of unique virtual writers supported by a **DDS_DataWriter** (p. 485), 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. 1326)

[range] [1, 1000000], or **DDS_LENGTH_UNLIMITED** (p. 1326)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.41.2.11 max_remote_readers

DDS_Long DDS_DataWriterResourceLimitsQosPolicy::max_remote_readers

<<**extension**>> (p. 795) The maximum number of remote readers supported by a **DDS_DataWriter** (p. 485)

[default] **DDS_LENGTH_UNLIMITED** (p. 1326)

[range] [1, 1000000], or **DDS_LENGTH_UNLIMITED** (p. 1326)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.41.2.12 max_app_ack_remote_readers

DDS_Long DDS_DataWriterResourceLimitsQosPolicy::max_app_ack_remote_readers

<<**extension**>> (p. 795) The maximum number of application-level acknowledging remote readers supported by a **DDS_DataWriter** (p. 485)

[default] **DDS_LENGTH_UNLIMITED** (p. 1326)

[range] [1, 1000000], or **DDS_LENGTH_UNLIMITED** (p. 1326)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Application Acknowledgement** (p. 835), **Reliability** (p. 895)

6.41.2.13 initial_active_topic_queries

DDS_Long DDS_DataWriterResourceLimitsQosPolicy::initial_active_topic_queries

<<**extension**>> (p. 795) Represents the initial number of active topic queries a **DDS_DataWriter** (p. 485) will manage.

[default] 1

[range] [1, 1000000]

See also

DDS_TopicQueryDispatchQosPolicy (p. 2047)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Topic Queries** (p. 911)

6.41.2.14 max_active_topic_queries

DDS_Long DDS_DataWriterResourceLimitsQosPolicy::max_active_topic_queries

<<**extension**>> (p. 795) Represents the maximum number of active topic queries a **DDS_DataWriter** (p. 485) will manage.

When topic queries are enabled, the maximum number of topic queries that a **DDS_DataWriter** (p. 485) 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. 1326)

[range] [1, 1000000] or DDS_LENGTH_UNLIMITED (p. 1326)

See also

DDS_TopicQueryDispatchQosPolicy (p. 2047)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Topic Queries** (p. 911)

6.41.2.15 writer_loaned_sample_allocation

struct DDS_AllocationSettings_t DDS_DataWriterResourceLimitsQosPolicy::writer_loaned_sample_allocation

<<**extension**>> (p. 795) Represents the allocation settings of loaned samples managed by a **DDS_DataWriter** (p. 485).

The number of samples loaned by a **DDS_DataWriter** (p. 485) via **FooDataWriter_get_loan** (p. 508) is limited by the **DDS_AllocationSettings_t::max_count** (p. 1538) of **DDS_DataWriterResourceLimitsQosPolicy::writer_loaned_sample_allocation** (p. 1681). **FooDataWriter_get_loan** (p. 508) returns NULL if and only if **DDS_AllocationSettings_t::max_count** (p. 1538) samples have been loaned, and none of those samples has been written with **FooDataWriter_write** (p. 496) or discarded via **FooDataWriter_discard_loan** (p. 510).

[default] initial_count = DDS_AUTO_COUNT (p. 1284) (**DDS_ResourceLimitsQosPolicy::initial_samples** (p. 1952) + 1); max_count = DDS_AUTO_COUNT (p. 1284) (**DDS_ResourceLimitsQosPolicy::max_samples** (p. 1951) + 1); incremental_count = DDS_AUTO_COUNT (p. 1284) (0 if initial_count = max_count; initial_count otherwise);

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 1537)

See also

FooDataWriter_get_loan (p. 508)

FooDataWriter_discard_loan (p. 510)

Categories

Immutable (p. 914), **Sample Memory Management** (p. 902), **Middleware Memory Management** (p. 878), **Sending Large Data** (p. 871)

6.41.2.16 initialize_writer_loaned_sample

DDS_Boolean DDS_DataWriterResourceLimitsQosPolicy::initialize_writer_loaned_sample

<<**extension**>> (p. 795) Whether or not to initialize loaned samples returned by a **DDS_DataWriter** (p. 485)

[default] **DDS_BOOLEAN_FALSE** (p. 1201)

See also

FooDataWriter_get_loan (p. 508)

Categories

Immutable (p. 914), **Sample Memory Management** (p. 902), **Sending Large Data** (p. 871)

6.42 DDS_DataWriterShmemRefTransferModeSettings Struct Reference

Settings related to transferring data using shared memory references.

Data Fields

- **DDS_Boolean enable_data_consistency_check**
Controls if samples can be checked for consistency.

6.42.1 Detailed Description

Settings related to transferring data using shared memory references.

It is used to configure a **DDS_DataWriter** (p. 485) using **Zero Copy transfer over shared memory** (p. 210).

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

QoS:

DDS_DataWriterTransferModeQosPolicy (p. 1683)

6.42.2 Field Documentation

6.42.2.1 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 **DDS_DataWriter** (p. 485) sends an incrementing sequence number as an inline QoS with every sample. This sequence number allows a **DDS_DataReader** (p. 615) to use the **FooDataReader_is_data_consistent** (p. 649) API to detect if the **DDS_DataWriter** (p. 485) overwrote the sample before the **DDS_DataReader** (p. 615) could complete processing the sample.

[default] **DDS_BOOLEAN_TRUE** (p. 1201)

6.43 DDS_DataWriterTransferModeQosPolicy Struct Reference

<<*extension*>> (p. 795) Qos related to transferring data

Data Fields

- struct **DDS_DataWriterShmemRefTransferModeSettings** **shmem_ref_settings**
<<*extension*>> (p. 795) *Settings related to transferring data using shared memory references.*

6.43.1 Detailed Description

<<*extension*>> (p. 795) Qos related to transferring data

It contains qualitative settings related to the actions a **DDS_DataWriter** (p. 485) performs while transferring its data.

Entity:

DDS_DataWriter (p. 485)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

6.43.2 Field Documentation

6.43.2.1 shmem_ref_settings

```
struct DDS_DataWriterShmemRefTransferModeSettings DDS_DataWriterTransferModeQosPolicy::shmem_ref_settings
```

<<**extension**>> (p. 795) Settings related to transferring data using shared memory references.

For details, refer to the **DDS_DataWriterShmemRefTransferModeSettings** (p. 1682)

Categories

Immutable (p. 914), **Sending Large Data** (p. 871)

6.44 DDS_DeadlineQosPolicy Struct Reference

<<**spec**>> (p. 790) Expresses the maximum duration (deadline) within which an instance is expected to be updated.

Data Fields

- struct **DDS_Duration_t period**
<<**basic**>> (p. 783) <<**spec**>> (p. 790) *Duration of the deadline period.*

6.44.1 Detailed Description

<<**spec**>> (p. 790) Expresses the maximum duration (deadline) within which an instance is expected to be updated.

A **DDS_DataReader** (p. 615) 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 **DDS_DataWriter** (p. 485) indicates that the application commits to write a new value (using the **DDS_DataWriter** (p. 485)) for each instance managed by the **DDS_DataWriter** (p. 485) 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:

DDS_Topic (p. 175), **DDS_DataReader** (p. 615), **DDS_DataWriter** (p. 485)

Status:

DDS_OFFERED_DEADLINE_MISSED_STATUS (p. 1228), **DDS_REQUESTED_DEADLINE_MISSED_STATUS** (p. 1229), **DDS_OFFERED_INCOMPATIBLE_QOS_STATUS** (p. 1229), **DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 1229)

Properties:

RxO (p. 1245) = YES
Changeable (p. 1245) = **YES** (p. 1245)

6.44.2 Usage

This policy is useful for cases where a **DDS_Topic** (p. 175) 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 **DDS_DataWriter** (p. 485) and a **DDS_DataReader** (p. 615) it checks whether the settings are compatible (i.e., *offered deadline* $\leq$ *requested deadline*); if they are not, the two entities are informed (via the **DDS_Listener** (p. 1816) or **DDS_Condition** (p. 1370) 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 **DDS_Listener** (p. 1816) or **DDS_Condition** (p. 1370).

6.44.3 Compatibility

The value offered is considered compatible with the value requested if and only if the inequality *offered period* $\leq$ *requested period* holds.

6.44.4 Consistency

The setting of the **DEADLINE** (p. 1272) policy must be set consistently with that of the **TIME_BASED_FILTER** (p. 1329).

For these two policies to be consistent the settings must be such that *deadline period* $\geq$ *minimum_separation*.

An attempt to set these policies in an inconsistent manner will result in **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222) in **set_qos (abstract)** (p. 1361), or the **DDS_Entity** (p. 1360) will not be created.

For a **DDS_DataReader** (p. 615), the **DEADLINE** (p. 1272) policy and **DDS_TimeBasedFilterQosPolicy** (p. 2032) may interact such that even though the **DDS_DataWriter** (p. 485) is writing samples fast enough to fulfill its commitment to its own deadline, the **DDS_DataReader** (p. 615) may see violations of its deadline. This happens because RTI Connext will drop any samples received within the **DDS_TimeBasedFilterQosPolicy::minimum_separation** (p. 2034). To avoid triggering the **DDS_DataReader** (p. 615)'s deadline, even though the matched **DDS_DataWriter** (p. 485) is meeting its own deadline, set the two QoS parameters so that:

reader deadline $\geq$ *reader minimum_separation* + *writer deadline*

See **DDS_TimeBasedFilterQosPolicy** (p. 2032) for more information about the interactions between deadlines and time-based filters.

See also

DDS_TimeBasedFilterQosPolicy (p. 2032)

6.44.5 Field Documentation

6.44.5.1 period

```
struct DDS_Duration_t DDS_DeadlineQosPolicy::period
```

<<**basic**>> (p. 783) <<**spec**>> (p. 790) Duration of the deadline period.

[default] **DDS_DURATION_INFINITE** (p. 1208)

[range] [1 nanosec, 1 year] or **DDS_DURATION_INFINITE** (p. 1208), >= **DDS_TimeBasedFilterQosPolicy**↔
::minimum_separation (p. 2034)

Categories

Mutable (p. 949), **Discovery Metadata** (p. 958), **Matching** (p. 963), **Applied Per-Instance** (p. 956), **Data Availability** (p. 843)

6.45 DDS_DestinationOrderQosPolicy Struct Reference

<<**spec**>> (p. 790) Controls how the middleware will deal with data sent by multiple **DDS_DataWriter** (p. 485) entities for the same instance of data (i.e., same **DDS_Topic** (p. 175) and key).

Data Fields

- **DDS_DestinationOrderQosPolicyKind kind**
 <<**basic**>> (p. 783) <<**spec**>> (p. 790) Specifies the desired kind of destination order.
- **DDS_DestinationOrderQosPolicyScopeKind scope**
 <<**spec**>> (p. 790) Specifies the desired scope of the source destination order.
- struct **DDS_Duration_t source_timestamp_tolerance**
 <<**extension**>> (p. 795) Allowed tolerance between source timestamps of consecutive samples.

6.45.1 Detailed Description

<<**spec**>> (p. 790) Controls how the middleware will deal with data sent by multiple **DDS_DataWriter** (p. 485) entities for the same instance of data (i.e., same **DDS_Topic** (p. 175) and key).

Entity:

DDS_Topic (p. 175), **DDS_DataReader** (p. 615), **DDS_DataWriter** (p. 485)

Status:

DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 1229), **DDS_REQUESTED_INCOMPATIBLE_QOS**↔
STATUS (p. 1229)

Properties:

RxO (p. 1245) = YES
Changeable (p. 1245) = **UNTIL ENABLE** (p. 1245)

6.45.2 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 **DDS_DataWriter** (p. 485) entities (which may be associated with different **DDS_Publisher** (p. 443) 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. 1274), indicates that (assuming the **OWNERSHIP_STRENGTH** (p. 1302) 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 **DDS_DataReader** (p. 615) in the order in which it was *received* (which may lead to inconsistent final values).

For **DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 1274), if the scope is set to **DDS_↔INSTANCE_SCOPE_DESTINATIONORDER_QOS** (p. 1275) (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. 1275), 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. 1274), the DataWriter will not enforce source timestamp ordering when writing samples using the **FooDataWriter_write_w_params** (p. 501) or **FooDataWriter_write_w_timestamp** (p. 500) 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. 1274), If scope is set to **DDS_↔INSTANCE_SCOPE_DESTINATIONORDER_QOS** (p. 1275) (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. 1222).

If scope is set to **DDS_TOPIC_SCOPE_DESTINATIONORDER_QOS** (p. 1275), 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.)

6.45.3 Compatibility

The value offered is considered compatible with the value requested if and only if the inequality *offered kind* $\geq$ *requested kind* evaluates to 'TRUE'. For the purposes of this inequality, the values of **DDS_DestinationOrderQosPolicy**↵
::kind (p. 1688) are considered ordered such that **DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS**
(p. 1274) < **DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 1274)

6.45.4 Field Documentation

6.45.4.1 kind

DDS_DestinationOrderQosPolicyKind DDS_DestinationOrderQosPolicy::kind

<<**basic**>> (p. 783) <<**spec**>> (p. 790) Specifies the desired kind of destination order.

[default] **DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS** (p. 1274),

Categories

Immutable (p. 914), **Matching** (p. 963), **Discovery Metadata** (p. 958), **Applied Per-Instance** (p. 956), **Sample Ordering** (p. 905)

6.45.4.2 scope

DDS_DestinationOrderQosPolicyScopeKind DDS_DestinationOrderQosPolicy::scope

<<**spec**>> (p. 790) 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. 1275)

Categories

Immutable (p. 914), **Applied Per-Instance** (p. 956), **Sample Ordering** (p. 905)

6.45.4.3 source_timestamp_tolerance

```
struct DDS_Duration_t DDS_DestinationOrderQosPolicy::source_timestamp_tolerance
```

<<**extension**>> (p. 795) Allowed tolerance between source timestamps of consecutive samples.

When a **DDS_DataWriter** (p. 485) sets **DDS_DestinationOrderQosPolicyKind** (p. 1274) to **DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 1274), 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 **DDS_DataReader** (p. 615) sets **DDS_DestinationOrderQosPolicyKind** (p. 1274) to **DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS** (p. 1274), the **DDS_DataReader** (p. 615) 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 **DDS_DataWriter** (p. 485), 30 seconds for **DDS_DataReader** (p. 615)

Categories

Immutable (p. 914), **Applied Per-Instance** (p. 956), **Sample Ordering** (p. 905)

6.46 DDS_DiscoveryConfigQosPolicy Struct Reference

Settings for discovery configuration.

Data Fields

- struct **DDS_Duration_t participant_liveliness_lease_duration**
 <<**basic**>> (p. 783) <<**extension**>> (p. 795) *The liveliness lease duration for the participant.*
- struct **DDS_Duration_t participant_liveliness_assert_period**
 <<**basic**>> (p. 783) <<**extension**>> (p. 795) *The period to assert liveliness for the participant.*
- struct **DDS_Duration_t participant_announcement_period**
 <<**basic**>> (p. 783) <<**extension**>> (p. 795) *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. 965) <<**extension**>> (p. 795) *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. 965) <<**extension**>> (p. 795) *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. 783) <<**extension**>> (p. 795) *The number of initial announcements sent when a participant is first enabled.*
- **DDS_Long new_remote_participant_announcements**
 <<**basic**>> (p. 783) <<**extension**>> (p. 795) *The number of participant announcements sent when a remote participant is newly discovered.*

- struct **DDS_Duration_t min_initial_participant_announcement_period**
 <<**basic**>> (p. 783) <<**extension**>> (p. 795) *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. 783) <<**extension**>> (p. 795) *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. 795) *Resource limits.*
- struct **DDS_RtpsReliableReaderProtocol_t publication_reader**
 <<**extension**>> (p. 795) *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. 795) *Resource limits.*
- struct **DDS_RtpsReliableReaderProtocol_t subscription_reader**
 <<**extension**>> (p. 795) *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. 795) *Resource limits.*
- struct **DDS_RtpsReliableWriterProtocol_t publication_writer**
 <<**extension**>> (p. 795) *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. 795) *Writer data lifecycle settings for a built-in publication writer.*
- struct **DDS_RtpsReliableWriterProtocol_t subscription_writer**
 <<**extension**>> (p. 795) *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. 795) *Writer data lifecycle settings for a built-in subscription writer.*
- **DDS_DiscoveryConfigBuiltinPluginKindMask builtin_discovery_plugins**
 <<**extension**>> (p. 795) *Mask of built-in discovery plugin kinds.*
- **DDS_DiscoveryConfigBuiltinChannelKindMask enabled_builtin_channels**
 <<**extension**>> (p. 795) *The mask specifying which built-in channels should be enabled.*
- **DDS_ReliabilityQosPolicyKind participant_message_reader_reliability_kind**
 [DEPRECATED] (p. 965) <<**extension**>> (p. 795) *Reliability policy for a built-in participant message reader.*
- struct **DDS_RtpsReliableReaderProtocol_t participant_message_reader**
 <<**extension**>> (p. 795) *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. 1701) is set to **DDS_RELIABLE_RELIABILITY_QOS** (p. 1323).*
- struct **DDS_RtpsReliableWriterProtocol_t participant_message_writer**
 <<**extension**>> (p. 795) *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. 1323) **DDS_ReliabilityQosPolicyKind** (p. 1322).*
- struct **DDS_PublishModeQosPolicy publication_writer_publish_mode**
 <<**extension**>> (p. 795) *Publish mode policy for the built-in publication writer.*
- struct **DDS_PublishModeQosPolicy subscription_writer_publish_mode**
 <<**extension**>> (p. 795) *Publish mode policy for the built-in subscription writer.*
- struct **DDS_AsynchronousPublisherQosPolicy asynchronous_publisher**
 <<**extension**>> (p. 795) *Asynchronous publishing settings for the discovery **DDS_Publisher** (p. 443) and all entities that are created by it.*

- struct **DDS_Duration_t default_domain_announcement_period**
 <<extension>> (p. 795) *The period to announce a participant to the default domain 0.*
- **DDS_Boolean ignore_default_domain_announcements**
 [DEPRECATED] (p. 965) <<extension>> (p. 795) *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. 795) *RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in DDS_ServiceRequest (p. 2001) writer.*
- struct **DDS_WriterDataLifecycleQosPolicy service_request_writer_data_lifecycle**
 <<extension>> (p. 795) *Writer data lifecycle settings for a built-in DDS_ServiceRequest (p. 2001) writer.*
- struct **DDS_PublishModeQosPolicy service_request_writer_publish_mode**
 <<extension>> (p. 795) *Publish mode policy for the built-in service request writer.*
- struct **DDS_RtpsReliableReaderProtocol_t service_request_reader**
 <<extension>> (p. 795) *RTPS reliable reader protocol-related configuration settings for a built-in DDS_ServiceRequest (p. 2001) reader.*
- struct **DDS_Duration_t locator_reachability_assert_period**
 <<extension>> (p. 795) *Period at which this DomainParticipant will assert the locators discovered from other DomainParticipants.*
- struct **DDS_Duration_t locator_reachability_lease_duration**
 <<extension>> (p. 795) *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. 795) *Period at which this DomainParticipant will check if its locators are reachable from other DomainParticipants.*
- struct **DDS_RtpsReliableWriterProtocol_t secure_volatile_writer**
 <<extension>> (p. 795) *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. 795) *Publish mode policy for the built-in secure volatile writer.*
- struct **DDS_RtpsReliableReaderProtocol_t secure_volatile_reader**
 <<extension>> (p. 795) *RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.*
- **DDS_Long endpoint_type_object_lb_serialization_threshold**
 <<extension>> (p. 795) *Option to reduce the size required to propagate a TypeObject in Simple Endpoint Discovery*
- struct **DDS_Duration_t dns_tracker_polling_period**
 <<extension>> (p. 795) *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. 795) *Publish mode policy for the built-in participant configuration writer.*
- struct **DDS_RtpsReliableWriterProtocol_t participant_configuration_writer**
 <<extension>> (p. 795) *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. 795) *Writer data lifecycle settings for a built-in participant configuration writer.*
- struct **DDS_RtpsReliableReaderProtocol_t participant_configuration_reader**
 <<extension>> (p. 795) *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. 795) *Resource limits for the built-in topic participant configuration reader.*

6.46.1 Detailed Description

Settings for discovery configuration.

<<**extension**>> (p. 795) 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:

DDS_DomainParticipant (p. 77)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

6.46.2 Field Documentation

6.46.2.1 participant_liveliness_lease_duration

```
struct DDS_Duration_t DDS_DiscoveryConfigQosPolicy::participant_liveliness_lease_duration
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) 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. 1692).

[default] 100 seconds

[range] [1 nanosec, 1 year], > participant_liveliness_assert_period

Categories

Immutable (p. 914), **Discovery** (p. 853), **Liveliness** (p. 874), **Discovery Metadata** (p. 958)

6.46.2.2 participant_liveliness_assert_period

```
struct DDS_Duration_t DDS_DiscoveryConfigQosPolicy::participant_liveliness_assert_period
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) 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. 1692).

[default] 30 seconds

[range] [1 nanosec, 1 year], < participant_liveliness_lease_duration

Categories

Immutable (p. 914), **Discovery** (p. 853), **Liveliness** (p. 874)

6.46.2.3 participant_announcement_period

```
struct DDS_Duration_t DDS_DiscoveryConfigQosPolicy::participant_announcement_period
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) 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. 1716) list **DDS_DomainParticipant_add_peer** (p. 149) API are used to configure a set of potential peers that a DomainParticipant may discover. The **DDS_DiscoveryConfigQosPolicy::participant_announcement_period** (p. 1693) 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. 1209) (Takes the value of **DDS_DiscoveryConfigQosPolicy::participant_liveliness_assert_period** (p. 1692))

[range] [1 nanosec, 1 year]

Categories

Immutable (p. 914), **Discovery** (p. 853)

6.46.2.4 remote_participant_purge_kind

DDS_RemoteParticipantPurgeKind DDS_DiscoveryConfigQosPolicy::remote_participant_purge_kind

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) 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. 1282). However, **DDS_NO_REMOTE_PARTICIPANT_PURGE** (p. 1283) 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. 1296).

[default] **DDS_LIVELINESS_BASED_REMOTE_PARTICIPANT_PURGE** (p. 1282)

Categories

Immutable (p. 914), **Discovery** (p. 853), **Liveliness** (p. 874)

6.46.2.5 max_liveliness_loss_detection_period

struct DDS_Duration_t DDS_DiscoveryConfigQosPolicy::max_liveliness_loss_detection_period

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) 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 **DDS_DataReader** (p. 615) will learn of the loss of liveliness of a matched **DDS_DataWriter** (p. 485) 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. 914), **Discovery** (p. 853), **Liveliness** (p. 874)

6.46.2.6 initial_participant_announcements

DDS_Long DDS_DiscoveryConfigQosPolicy::initial_participant_announcements

<<**basic**>> (p. 783) <<**extension**>> (p. 795) The number of initial announcements sent when a participant is first enabled.

[default] 5

[range] [1,1 million]

Categories

Immutable (p. 914), **Discovery** (p. 853)

6.46.2.7 new_remote_participant_announcements

DDS_Long DDS_DiscoveryConfigQosPolicy::new_remote_participant_announcements

<<**basic**>> (p. 783) <<**extension**>> (p. 795) 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. 914), **Discovery** (p. 853)

6.46.2.8 min_initial_participant_announcement_period

struct DDS_Duration_t DDS_DiscoveryConfigQosPolicy::min_initial_participant_announcement_period

<<**basic**>> (p. 783) <<**extension**>> (p. 795) 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. 1695) is introduced in between initial announcements when a new remote participant is discovered.

The setting of **DDS_DiscoveryConfigQosPolicy::min_initial_participant_announcement_period** (p. 1695) must be consistent with **DDS_DiscoveryConfigQosPolicy::max_initial_participant_announcement_period** (p. 1695). For these two values to be consistent, they must verify that:

DDS_DiscoveryConfigQosPolicy::min_initial_participant_announcement_period (p. 1695) <= **DDS_DiscoveryConfigQosPolicy::max_initial_participant_announcement_period** (p. 1695).

[default] 10 milliseconds

[range] [1 nanosec,1 year]

Categories

Immutable (p. 914), **Discovery** (p. 853)

6.46.2.9 max_initial_participant_announcement_period

```
struct DDS_Duration_t DDS_DiscoveryConfigQosPolicy::max_initial_participant_announcement_period
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) 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. 1695) 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. 1695) must be consistent with **DDS_DiscoveryConfigQosPolicy::min_initial_participant_announcement_period** (p. 1695). For these two values to be consistent, they must verify that:

DDS_DiscoveryConfigQosPolicy::min_initial_participant_announcement_period (p. 1695) ≤ **DDS_DiscoveryConfigQosPolicy::max_initial_participant_announcement_period** (p. 1695).

[default] 1 second

[range] [1 nanosec, 1 year]

Categories

Immutable (p. 914), **Discovery** (p. 853)

6.46.2.10 participant_reader_resource_limits

```
struct DDS_BuiltinTopicReaderResourceLimits_t DDS_DiscoveryConfigQosPolicy::participant_reader_resource_limits
```

<<**extension**>> (p. 795) Resource limits.

Resource limit of the built-in topic participant reader. For details, see **DDS_BuiltinTopicReaderResourceLimits_t** (p. 1557).

Categories

Immutable (p. 914), **Discovery** (p. 853), **Sample Memory Management** (p. 902), **Middleware Memory Management** (p. 878)

6.46.2.11 publication_reader

```
struct DDS_RtpsReliableReaderProtocol_t DDS_DiscoveryConfigQosPolicy::publication_reader
```

<<**extension**>> (p. 795) 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. 1954)

[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. 914), **Discovery** (p. 853), **Reliability** (p. 895)

6.46.2.12 publication_reader_resource_limits

```
struct DDS_BuiltinTopicReaderResourceLimits_t DDS_DiscoveryConfigQosPolicy::publication_reader_↔  
resource_limits
```

<<**extension**>> (p. 795) Resource limits.

Resource limit of the built-in topic publication reader. For details, see **DDS_BuiltinTopicReaderResourceLimits_t** (p. 1557).

Categories

Immutable (p. 914), **Discovery** (p. 853), **Sample Memory Management** (p. 902), **Middleware Memory Management** (p. 878)

6.46.2.13 subscription_reader

```
struct DDS_RtpsReliableReaderProtocol_t DDS_DiscoveryConfigQosPolicy::subscription_reader
```

<<**extension**>> (p. 795) 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. 1954)

[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. 914), **Discovery** (p. 853), **Reliability** (p. 895)

6.46.2.14 subscription_reader_resource_limits

```
struct DDS_BuiltinTopicReaderResourceLimits_t DDS_DiscoveryConfigQosPolicy::subscription_reader↵
_resource_limits
```

<<**extension**>> (p. 795) Resource limits.

Resource limit of the built-in topic subscription reader. For details, see **DDS_BuiltinTopicReaderResourceLimits_t** (p. 1557).

Categories

Immutable (p. 914), **Discovery** (p. 853), **Sample Memory Management** (p. 902), **Middleware Memory Management** (p. 878)

6.46.2.15 publication_writer

```
struct DDS_RtpsReliableWriterProtocol_t DDS_DiscoveryConfigQosPolicy::publication_writer
```

<<**extension**>> (p. 795) 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. 1960)

[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. 1208);
samples_per_virtual_heartbeat DDS_LENGTH_UNLIMITED (p. 1326);
max_heartbeat_retries 10;
inactivate_nonprogressing_readers DDS_BOOLEAN_FALSE (p. 1201);
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. 1201);
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. 1326);
max_send_window_size DDS_LENGTH_UNLIMITED (p. 1326);
send_window_update_period 3s;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat DDS_BOOLEAN_FALSE (p. 1201);
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat DDS_BOOLEAN_FALSE (p. 1201);
```

Categories

Immutable (p. 914), **Discovery** (p. 853), **Reliability** (p. 895)

6.46.2.16 publication_writer_data_lifecycle

```
struct DDS_WriterDataLifecycleQosPolicy DDS_DiscoveryConfigQosPolicy::publication_writer_data_↵
lifecycle
```

<<**extension**>> (p. 795) Writer data lifecycle settings for a built-in publication writer.

For details, refer to the **DDS_WriterDataLifecycleQosPolicy** (p. 2106). **DDS_WriterDataLifecycleQosPolicy**↵
::autodispose_unregistered_instances (p. 2107) will always be forced to **DDS_BOOLEAN_TRUE** (p. 1201).

Categories

Immutable (p. 914), **Discovery** (p. 853), **Instance Lifecycle** (p. 867)

6.46.2.17 subscription_writer

```
struct DDS_RtpsReliableWriterProtocol_t DDS_DiscoveryConfigQosPolicy::subscription_writer
```

<<**extension**>> (p. 795) 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. 1960)

[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. 1208);
samples_per_virtual_heartbeat DDS_LENGTH_UNLIMITED (p. 1326);
max_heartbeat_retries 10;
inactivate_nonprogressing_readers DDS_BOOLEAN_FALSE (p. 1201);
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. 1201);
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. 1326);
max_send_window_size DDS_LENGTH_UNLIMITED (p. 1326);
send_window_update_period 3s;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat DDS_BOOLEAN_FALSE (p. 1201);
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat DDS_BOOLEAN_FALSE (p. 1201);
```

Categories

Immutable (p. 914), **Discovery** (p. 853), **Reliability** (p. 895)

6.46.2.18 subscription_writer_data_lifecycle

```
struct DDS_WriterDataLifecycleQosPolicy DDS_DiscoveryConfigQosPolicy::subscription_writer_data_↵
lifecycle
```

<<**extension**>> (p. 795) Writer data lifecycle settings for a built-in subscription writer.

For details, refer to the **DDS_WriterDataLifecycleQosPolicy** (p. 2106). **DDS_WriterDataLifecycleQosPolicy**↵
::autodispose_unregistered_instances (p. 2107) will always be forced to **DDS_BOOLEAN_TRUE** (p. 1201).

Categories

Immutable (p. 914), **Discovery** (p. 853), **Instance Lifecycle** (p. 867)

6.46.2.19 builtin_discovery_plugins

```
DDS_DiscoveryConfigBuiltinPluginKindMask DDS_DiscoveryConfigQosPolicy::builtin_discovery_plugins
```

<<**extension**>> (p. 795) 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. 1280)

See also

DDS_DiscoveryConfigBuiltinPluginKind (p. 1280)

Categories

Immutable (p. 914), **Discovery** (p. 853)

6.46.2.20 enabled_builtin_channels

```
DDS_DiscoveryConfigBuiltinChannelKindMask DDS_DiscoveryConfigQosPolicy::enabled_builtin_channels
```

<<**extension**>> (p. 795) The mask specifying which built-in channels should be enabled.

While there are a number of built-in channels that are used by Connex DDS, the only built-in channel which can currently be enabled or disabled is the Service Request Channel. This channel is used by the Locator Reachability, Instance State Consistency and Topic Query features. If you are not using these features and wish to reduce network traffic and endpoint resource usage, you may disable the service request channel with this QoS. [default] **DDS**↵
DISCOVERYCONFIG_SERVICE_REQUEST_CHANNEL (p. 1281)

Categories

Immutable (p. 914)

6.46.2.21 participant_message_reader_reliability_kind

DDS_ReliabilityQosPolicyKind `DDS_DiscoveryConfigQosPolicy::participant_message_reader_reliability_↵_kind`

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Reliability policy for a built-in participant message reader.

For details, refer to the **DDS_ReliabilityQosPolicyKind** (p. 1322).

[default] **DDS_BEST_EFFORT_RELIABILITY_QOS** (p. 1323)

Categories

Immutable (p. 914), **Discovery** (p. 853), **Reliability** (p. 895), **Liveliness** (p. 874)

6.46.2.22 participant_message_reader

struct DDS_RtpsReliableReaderProtocol_t `DDS_DiscoveryConfigQosPolicy::participant_message_reader`

<<*extension*>> (p. 795) 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. 1701) is set to **DDS_RELIABLE_RELIABILITY_QOS** (p. 1323).

For details, refer to the **DDS_RtpsReliableReaderProtocol_t** (p. 1954)

[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. 914), **Discovery** (p. 853), **Reliability** (p. 895), **Liveliness** (p. 874)

6.46.2.23 participant_message_writer

```
struct DDS_RtpsReliableWriterProtocol_t DDS_DiscoveryConfigQosPolicy::participant_message_writer
```

<<**extension**>> (p. 795) 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. 1323) **DDS_ReliabilityQosPolicyKind** (p. 1322).

For details, refer to the **DDS_RtpsReliableWriterProtocol_t** (p. 1960)

[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. 1208);
samples_per_virtual_heartbeat DDS_LENGTH_UNLIMITED (p. 1326);
max_heartbeat_retries 10;
inactivate_nonprogressing_readers DDS_BOOLEAN_FALSE (p. 1201);
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. 1201);
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. 1326);
max_send_window_size DDS_LENGTH_UNLIMITED (p. 1326);
send_window_update_period 1s;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat DDS_BOOLEAN_FALSE (p. 1201);
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat DDS_BOOLEAN_FALSE (p. 1201);
```

Categories

Immutable (p. 914), **Discovery** (p. 853), **Reliability** (p. 895), **Liveliness** (p. 874)

6.46.2.24 publication_writer_publish_mode

```
struct DDS_PublishModeQosPolicy DDS_DiscoveryConfigQosPolicy::publication_writer_publish_mode
```

<<**extension**>> (p. 795) Publish mode policy for the built-in publication writer.

Determines whether the Discovery built-in publication **DDS_DataWriter** (p. 485) publishes data synchronously or asynchronously and how.

Categories

Immutable (p. 914), **Discovery** (p. 853), **Flow Control** (p. 863)

6.46.2.25 subscription_writer_publish_mode

```
struct DDS_PublishModeQosPolicy DDS_DiscoveryConfigQosPolicy::subscription_writer_publish_mode
```

<<**extension**>> (p. 795) Publish mode policy for the built-in subscription writer.

Determines whether the Discovery built-in subscription **DDS_DataWriter** (p. 485) publishes data synchronously or asynchronously and how.

Categories

Immutable (p. 914), **Discovery** (p. 853), **Flow Control** (p. 863)

6.46.2.26 asynchronous_publisher

```
struct DDS_AsynchronousPublisherQosPolicy DDS_DiscoveryConfigQosPolicy::asynchronous_publisher
```

<<**extension**>> (p. 795) Asynchronous publishing settings for the discovery **DDS_Publisher** (p. 443) and all entities that are created by it.

Categories

Immutable (p. 914), **Discovery** (p. 853), **Flow Control** (p. 863)

6.46.2.27 default_domain_announcement_period

```
struct DDS_Duration_t DDS_DiscoveryConfigQosPolicy::default_domain_announcement_period
```

<<**extension**>> (p. 795) 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. 1208).

When this period is set to a value other than **DDS_DURATION_INFINITE** (p. 1208) and **DDS_DiscoveryConfigQosPolicy::ignore_default_domain_announcements** (p. 1705) is set to **DDS_BOOLEAN_FALSE** (p. 1201), 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. 1870) 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. 1873).

[default] 30 seconds

[range] [1 nanosec, 1 year] or **DDS_DURATION_INFINITE** (p. 1208)

See also

DDS_ParticipantBuiltinTopicData::domain_id (p. 1873)

DDS_DiscoveryConfigQosPolicy::ignore_default_domain_announcements (p. 1705)

Categories

Immutable (p. 914), **Discovery** (p. 853), **Debuggability** (p. 847)

6.46.2.28 ignore_default_domain_announcements

```
DDS_Boolean DDS_DiscoveryConfigQosPolicy::ignore_default_domain_announcements
```

[DEPRECATED] (p. 965) <<**extension**>> (p. 795) 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. 1201), 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. 1201), a participant running on the default domain 0 will provide announcements from participants running on different domain IDs to the application via the **DDS_ParticipantBuiltinTopicData** (p. 1870) built-in DataReader.

[default] **DDS_BOOLEAN_TRUE** (p. 1201)

See also

DDS_ParticipantBuiltinTopicData::domain_id (p. 1873)

DDS_DiscoveryConfigQosPolicy::default_domain_announcement_period (p. 1704)

Categories

Immutable (p. 914), **Discovery** (p. 853), **Debuggability** (p. 847)

6.46.2.29 service_request_writer

```
struct DDS_RtpsReliableWriterProtocol_t DDS_DiscoveryConfigQosPolicy::service_request_writer
```

<<**extension**>> (p. 795) RTPS protocol-related configuration settings for the RTPS reliable writer associated with the built-in **DDS_ServiceRequest** (p. 2001) writer.

For details, refer to the **DDS_RtpsReliableWriterProtocol_t** (p. 1960)

[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. 1208);
samples_per_virtual_heartbeat DDS_LENGTH_UNLIMITED (p. 1326);
max_heartbeat_retries 10;
inactivate_nonprogressing_readers DDS_BOOLEAN_FALSE (p. 1201);
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. 1201);
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. 1326);
max_send_window_size DDS_LENGTH_UNLIMITED (p. 1326);
send_window_update_period 3s;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat DDS_BOOLEAN_FALSE (p. 1201);
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat DDS_BOOLEAN_FALSE (p. 1201);
```

Categories

Immutable (p. 914), **Discovery** (p. 853), **Reliability** (p. 895)

6.46.2.30 service_request_writer_data_lifecycle

```
struct DDS_WriterDataLifecycleQosPolicy DDS_DiscoveryConfigQosPolicy::service_request_writer_↔
data_lifecycle
```

<<**extension**>> (p. 795) Writer data lifecycle settings for a built-in **DDS_ServiceRequest** (p. 2001) writer.

For details, refer to the **DDS_WriterDataLifecycleQosPolicy** (p. 2106).

Categories

Immutable (p. 914), **Discovery** (p. 853), **Instance Lifecycle** (p. 867)

6.46.2.31 service_request_writer_publish_mode

```
struct DDS_PublishModeQosPolicy DDS_DiscoveryConfigQosPolicy::service_request_writer_publish_mode
```

<<**extension**>> (p. 795) Publish mode policy for the built-in service request writer.

Determines whether the Discovery built-in service request **DDS_DataWriter** (p. 485) publishes data synchronously or asynchronously and how.

Categories

Immutable (p. 914), **Discovery** (p. 853), **Flow Control** (p. 863)

6.46.2.32 service_request_reader

```
struct DDS_RtpsReliableReaderProtocol_t DDS_DiscoveryConfigQosPolicy::service_request_reader
```

<<**extension**>> (p. 795) RTPS reliable reader protocol-related configuration settings for a built-in **DDS_Service←Request** (p. 2001) reader.

For details, refer to the **DDS_RtpsReliableReaderProtocol_t** (p. 1954)

[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. 914), **Discovery** (p. 853), **Reliability** (p. 895)

6.46.2.33 locator_reachability_assert_period

```
struct DDS_Duration_t DDS_DiscoveryConfigQosPolicy::locator_reachability_assert_period
```

<<**extension**>> (p. 795) Period at which this DomainParticipant will assert the locators discovered from other DomainParticipants.

This setting configures the period at which this **DDS_DomainParticipant** (p. 77) 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. 1708).

If **DDS_DiscoveryConfigQosPolicy::locator_reachability_lease_duration** (p. 1708) is **DDS_DURATION_INFINITE** (p. 1208) 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. 1708)

Categories

Immutable (p. 914), **Transports** (p. 908)

6.46.2.34 locator_reachability_lease_duration

```
struct DDS_Duration_t DDS_DiscoveryConfigQosPolicy::locator_reachability_lease_duration
```

<<**extension**>> (p. 795) 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. 1208), 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. 1208)

[range] [1 nanosec, 1 year] or **DDS_DURATION_INFINITE** (p. 1208)

6.46.2.35 locator_reachability_change_detection_period

```
struct DDS_Duration_t DDS_DiscoveryConfigQosPolicy::locator_reachability_change_detection_period
```

<<**extension**>> (p. 795) 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. 1708) value.

If **DDS_DiscoveryConfigQosPolicy::locator_reachability_lease_duration** (p. 1708) is **DDS_DURATION_INFINITE** (p. 1208) 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. 1708)

Categories

Immutable (p. 914), **Transports** (p. 908)

6.46.2.36 secure_volatile_writer

```
struct DDS_RtpsReliableWriterProtocol_t DDS_DiscoveryConfigQosPolicy::secure_volatile_writer
```

<<**extension**>> (p. 795) 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. 1960)

[default]

```
low_watermark 0;
high_watermark 1;
heartbeat_period 1.0 second;
fast_heartbeat_period 250.0 milliseconds;
late_joiner_heartbeat_period 1.0 second;
virtual_heartbeat_period DDS_DURATION_INFINITE (p. 1208);
samples_per_virtual_heartbeat DDS_LENGTH_UNLIMITED (p. 1326);
max_heartbeat_retries 800;
inactivate_nonprogressing_readers DDS_BOOLEAN_FALSE (p. 1201);
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 millisecond;
disable_positive_acks_max_sample_keep_duration 1.0 second;
disable_positive_acks_enable_adaptive_sample_keep_duration DDS_BOOLEAN_TRUE (p. 1201);
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. 1326);
max_send_window_size DDS_LENGTH_UNLIMITED (p. 1326);
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. 1201);
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat DDS_BOOLEAN_FALSE (p. 1201);

```

Categories

Immutable (p. 914), **Discovery** (p. 853), **Reliability** (p. 895), **Security** (p. 906)

6.46.2.37 secure_volatile_writer_publish_mode

```
struct DDS_PublishModeQosPolicy DDS_DiscoveryConfigQosPolicy::secure_volatile_writer_publish_mode
```

<<**extension**>> (p. 795) Publish mode policy for the built-in secure volatile writer.

Determines whether the built-in secure volatile **DDS_DataWriter** (p. 485) publishes data synchronously or asynchronously and how.

Categories

Immutable (p. 914), **Discovery** (p. 853), **Flow Control** (p. 863), **Security** (p. 906)

6.46.2.38 secure_volatile_reader

```
struct DDS_RtpsReliableReaderProtocol_t DDS_DiscoveryConfigQosPolicy::secure_volatile_reader
```

<<**extension**>> (p. 795) RTPS reliable reader protocol-related configuration settings for the built-in secure volatile reader.

For details, refer to the **DDS_RtpsReliableReaderProtocol_t** (p. 1954)

[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. 914), **Discovery** (p. 853), **Reliability** (p. 895), **Security** (p. 906)

6.46.2.39 endpoint_type_object_lb_serialization_threshold

DDS_Long DDS_DiscoveryConfigQosPolicy::endpoint_type_object_lb_serialization_threshold

<<**extension**>> (p. 795) Option to reduce the size required to propagate a TypeObject in Simple Endpoint Discovery

Minimum size (in bytes) of the serialized TypeObject that will trigger the serialization of a TypeObjectLb instead of the regular TypeObject.

For example, setting this property to 1000 will trigger the serialization of the TypeObjectLb for TypeObjects whose serialized size is greater than 1000 Bytes.

The sentinel value -1 disables TypeObject compression.

[default] 0. The default value 0 enables TypeObject compression by always sending TypeObjectLb.

[range] [-1, 2147483647]

Categories

Immutable (p. 914), **Discovery** (p. 853), **Sending Large Data** (p. 871), **Type System** (p. 912)

6.46.2.40 dns_tracker_polling_period

struct DDS_Duration_t DDS_DiscoveryConfigQosPolicy::dns_tracker_polling_period

<<**extension**>> (p. 795) 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_DiscoveryQosPolicy**↔**::initial_peers** (p. 1716). The information regarding peers added through the **DDS_DomainParticipant_add_peer** (p. 149) 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. 1716) 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. 1692) (if using SPDP) or **DDS_DiscoveryConfigQosPolicy::participant_liveliness_assert_period** (p. 1692) (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. 1208), the tracker is disabled.

[default] 20 seconds

[range] [1 second, 1 year], **DDS_DURATION_INFINITE** (p. 1208)

Categories

Immutable (p. 914), **Discovery** (p. 853), **Transports** (p. 908)

6.46.2.41 participant_configuration_writer_publish_mode

```
struct DDS_PublishModeQosPolicy DDS_DiscoveryConfigQosPolicy::participant_configuration_writer_↵
publish_mode
```

<<**extension**>> (p. 795) Publish mode policy for the built-in participant configuration writer.

Determines whether the Discovery built-in participant configuration **DDS_DataWriter** (p. 485) publishes data synchronously or asynchronously and how.

Categories

Immutable (p. 914), **Discovery** (p. 853), **Flow Control** (p. 863)

6.46.2.42 participant_configuration_writer

```
struct DDS_RtpsReliableWriterProtocol_t DDS_DiscoveryConfigQosPolicy::participant_configuration←
_writer
```

<<**extension**>> (p. 795) 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. 1960)

[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. 1208);
samples_per_virtual_heartbeat DDS_LENGTH_UNLIMITED (p. 1326);
max_heartbeat_retries 10;
inactivate_nonprogressing_readers DDS_BOOLEAN_FALSE (p. 1201);
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. 1201);
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. 1326);
max_send_window_size DDS_LENGTH_UNLIMITED (p. 1326);
send_window_update_period 3s;
send_window_increase_factor 105;
send_window_decrease_factor 50;
enable_multicast_periodic_heartbeat DDS_BOOLEAN_FALSE (p. 1201);
multicast_resend_threshold 2 readers;
disable_repair_piggyback_heartbeat DDS_BOOLEAN_FALSE (p. 1201);
```

Categories

Immutable (p. 914), **Discovery** (p. 853), **Reliability** (p. 895)

6.46.2.43 participant_configuration_writer_data_lifecycle

```
struct DDS_WriterDataLifecycleQosPolicy DDS_DiscoveryConfigQosPolicy::participant_configuration←
_writer_data_lifecycle
```

<<**extension**>> (p. 795) Writer data lifecycle settings for a built-in participant configuration writer.

For details, refer to the **DDS_WriterDataLifecycleQosPolicy** (p. 2106). **DDS_WriterDataLifecycleQosPolicy**←
::autodispose_unregistered_instances (p. 2107) will always be forced to **DDS_BOOLEAN_TRUE** (p. 1201).

Categories

Immutable (p. 914), **Discovery** (p. 853), **Instance Lifecycle** (p. 867)

6.46.2.44 participant_configuration_reader

```
struct DDS_RtpsReliableReaderProtocol_t DDS_DiscoveryConfigQosPolicy::participant_configuration↔
_reader
```

<<**extension**>> (p. 795) 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. 1954)

[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. 914), **Discovery** (p. 853), **Reliability** (p. 895)

6.46.2.45 participant_configuration_reader_resource_limits

```
struct DDS_BuiltinTopicReaderResourceLimits_t DDS_DiscoveryConfigQosPolicy::participant_configuration↔
_reader_resource_limits
```

<<**extension**>> (p. 795) Resource limits for the built-in topic participant configuration reader.

For details, see **DDS_BuiltinTopicReaderResourceLimits_t** (p. 1557).

Categories

Immutable (p. 914), **Discovery** (p. 853), **Sample Memory Management** (p. 902), **Middleware Memory Management** (p. 878)

6.47 DDS_DiscoveryQosPolicy Struct Reference

<<**extension**>> (p. 795) Configures the mechanism used by the middleware to automatically discover and connect with new remote applications.

Data Fields

- struct **DDS_StringSeq enabled_transports**
 <<**extension**>> (p. 795) *The transports available for use by the Discovery mechanism.*
- struct **DDS_StringSeq initial_peers**
 <<**basic**>> (p. 783) <<**extension**>> (p. 795) *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. 783) <<**extension**>> (p. 795) *Specifies the multicast group addresses on which discovery-related meta-traffic can be received by the DomainParticipant.*
- **DDS_Long metatraffic_transport_priority**
 <<**extension**>> (p. 795) *The transport priority to use for the Discovery meta-traffic.*
- **DDS_Boolean accept_unknown_peers**
 <<**basic**>> (p. 783) <<**extension**>> (p. 795) *Whether to accept a new participant that is not in the initial peers list.*
- **DDS_Boolean enable_endpoint_discovery**
 [DEPRECATED] (p. 965) <<**extension**>> (p. 795) *Whether to automatically enable endpoint discovery for all the remote participants.*

6.47.1 Detailed Description

<<**extension**>> (p. 795) Configures the mechanism used by the middleware to automatically discover and connect with new remote applications.

Entity:

DDS_DomainParticipant (p. 77)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

6.47.2 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. 1352)

DDS_DiscoveryConfigQosPolicy (p. 1689)

6.47.3 Field Documentation

6.47.3.1 enabled_transports

```
struct DDS_StringSeq DDS_DiscoveryQosPolicy::enabled_transports
```

<<**extension**>> (p. 795) 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 **DDS_DataReader** (p. 615) and **DDS_DataWriter** (p. 485)).

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. 1527). In particular, be careful to avoid a situation in which RTI Connexx allocates a string on your behalf and you then reuse that string in such a way that RTI Connexx 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. 1331). 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. 914), **Transports** (p. 908), **Discovery** (p. 853)

6.47.3.2 initial_peers

```
struct DDS_StringSeq DDS_DiscoveryQosPolicy::initial_peers
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) 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 **DDS_DomainParticipant** (p. 77) 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 **DDS_DomainParticipant** (p. 77) is discovered by receiving participant announcements from a remote peer. When the new remote **DDS_DomainParticipant** (p. 77) 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. 1353)).

[default] builtin.udpv4://239.255.0.1, builtin.udpv4://127.0.0.1, builtin.shmem:// (See also **NDDS_DISCOVERY_PEERS** (p. 1352))

[range] Sequence of arbitrary length.

See also

Peer Descriptor Format (p. 1353)

DDS_DomainParticipant_add_peer() (p. 149)

Categories

Immutable (p. 914), **Transports** (p. 908), **Discovery Isolation** (p. 858), **Discovery** (p. 853)

6.47.3.3 multicast_receive_addresses

```
struct DDS_StringSeq DDS_DiscoveryQosPolicy::multicast_receive_addresses
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) 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. 1354)).

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_DiscoveryQosPolicy::multicast_receive_addresses** (p. 1717) 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. 1717), 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. 1717).

[default] builtin.udpv4://239.255.0.1 (See also **NDDS_DISCOVERY_PEERS** (p. 1352))

[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. 1354)

6.47.3.4 metatraffic_transport_priority

DDS_Long DDS_DiscoveryQosPolicy::metatraffic_transport_priority

<<**extension**>> (p. 795) The transport priority to use for the Discovery meta-traffic.

The discovery metatraffic will be sent by the built-in **DDS_DataWriter** (p. 485) using this transport priority.

[default] 0

[range] [0, MAX_UINT]

Categories

Immutable (p. 914), **Transports** (p. 908), **Concurrency** (p. 839), **Discovery** (p. 853)

6.47.3.5 accept_unknown_peers

DDS_Boolean DDS_DiscoveryQosPolicy::accept_unknown_peers

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Whether to accept a new participant that is not in the initial peers list.

If **DDS_BOOLEAN_FALSE** (p. 1201), the participant will only communicate with those in the initial peers list and those added via **DDS_DomainParticipant_add_peer()** (p. 149).

If **DDS_BOOLEAN_TRUE** (p. 1201), the participant will also communicate with all discovered remote participants.

Note: If `accept_unknown_peers` is **DDS_BOOLEAN_FALSE** (p. 1201) 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. 1352).

[default] **DDS_BOOLEAN_TRUE** (p. 1201)

Categories

Immutable (p. 914), **Transports** (p. 908), **Discovery Isolation** (p. 858), **Discovery** (p. 853)

6.47.3.6 enable_endpoint_discovery

DDS_Boolean DDS_DiscoveryQosPolicy::enable_endpoint_discovery

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Whether to automatically enable endpoint discovery for all the remote participants.

If **DDS_BOOLEAN_TRUE** (p. 1201), endpoint discovery will automatically occur for every discovered remote participant.

If **DDS_BOOLEAN_FALSE** (p. 1201), endpoint discovery will be initially disabled and manual activation is required for each discovered participant by calling **DDS_DomainParticipant_resume_endpoint_discovery** (p. 141).

[default] **DDS_BOOLEAN_TRUE** (p. 1201)

Categories

Immutable (p. 914), **Transports** (p. 908), **Discovery Isolation** (p. 858), **Discovery** (p. 853)

6.48 DDS_DomainParticipantConfigParams_t Struct Reference

<<*extension*>> (p. 1012) 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.

Data Fields

- int **domain_id**
*Domain ID from which the **DDS_DomainParticipant** (p. 77) is created.*
- char * **participant_name**
*Name assigned to the **DDS_DomainParticipant** (p. 77).*
- char * **participant_qos_library_name**
*QoS library name containing the QoS profile from which the **DDS_DomainParticipant** (p. 77) is created.*
- char * **participant_qos_profile_name**
*QoS profile name from which the **DDS_DomainParticipant** (p. 77) is created.*
- char * **domain_entity_qos_library_name**
*QoS library name containing the QoS profile from which the all the **DDS_DomainEntity** (p. 1363) defined under the participant configuration are created.*
- char * **domain_entity_qos_profile_name**
*QoS profile name from which the all the **DDS_DomainEntity** (p. 1363) defined under the participant configuration are created.*

6.48.1 Detailed Description

<<*extension*>> (p. 1012) 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.

6.48.2 Field Documentation

6.48.2.1 domain_id

```
int DDS_DomainParticipantConfigParams_t::domain_id
```

Domain ID from which the **DDS_DomainParticipant** (p. 77) 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. 1431) is specified then the ID in the configuration will be used.

6.48.2.2 participant_name

```
char* DDS_DomainParticipantConfigParams_t::participant_name
```

Name assigned to the **DDS_DomainParticipant** (p. 77).

This is the name that will be set in the **DDS_DomainParticipantQos::participant_name** (p. 1730). 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. 1431) then an automatically generated name will be assigned.

6.48.2.3 participant_qos_library_name

```
char* DDS_DomainParticipantConfigParams_t::participant_qos_library_name
```

QoS library name containing the QoS profile from which the **DDS_DomainParticipant** (p. 77) is created.

Allows overriding the QoS defined in the configuration for the participant to be created. This value only affects to the **DDS_DomainParticipant** (p. 77).

When this member is lexicographically equal to the special value **DDS_QOS_ELEMENT_NAME_USE_XML_CONFIG** (p. 1431) 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. 1720) will be ignored.

6.48.2.4 participant_qos_profile_name

```
char* DDS_DomainParticipantConfigParams_t::participant_qos_profile_name
```

QoS profile name from which the **DDS_DomainParticipant** (p. 77) is created.

Allows overriding the QoS defined in the configuration for the participant to be created. This value only affects to the **DDS_DomainParticipant** (p. 77).

When this member is lexicographically equal to the special value **DDS_QOS_ELEMENT_NAME_USE_XML_CONFIG** (p. 1431) 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. 1720) will be ignored.

6.48.2.5 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 **DDS_DomainEntity** (p. 1363) 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 **DDS_DomainEntity** (p. 1363).

When this member is lexicographically equal to the special value **DDS_QOS_ELEMENT_NAME_USE_XML_CONFIG** (p. 1431) 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. 1721) will be ignored.

6.48.2.6 domain_entity_qos_profile_name

```
char* DDS_DomainParticipantConfigParams_t::domain_entity_qos_profile_name
```

QoS profile name from which the all the **DDS_DomainEntity** (p. 1363) 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 **DDS_DomainEntity** (p. 1363).

When this member is lexicographically equal to the special value **DDS_QOS_ELEMENT_NAME_USE_XML_CONFIG** (p. 1431) 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. 1720) will be ignored.

6.49 DDS_DomainParticipantFactoryQos Struct Reference

QoS policies supported by a **DDS_DomainParticipantFactory** (p. 32).

Data Fields

- struct **DDS_EntityFactoryQosPolicy** `entity_factory`
Entity factory policy, **ENTITY_FACTORY** (p. 1289).
- struct **DDS_SystemResourceLimitsQosPolicy** `resource_limits`
<<extension>> (p. 1012) System resource limits, **SYSTEM_RESOURCE_LIMITS** (p. 1328).
- struct **DDS_ProfileQosPolicy** `profile`
<<extension>> (p. 1012) Qos profile policy, **PROFILE** (p. 1305).
- struct **DDS_LoggingQosPolicy** `logging`
<<extension>> (p. 1012) Logging qos policy, **LOGGING** (p. 1298).
- struct **DDS_MonitoringQosPolicy** `monitoring`
<<extension>> (p. 1012) Monitoring qos policy, **MONITORING** (p. 1299).

6.49.1 Detailed Description

QoS policies supported by a **DDS_DomainParticipantFactory** (p. 32).

Entity:

DDS_DomainParticipantFactory (p. 32)

See also

All QoS Policies (p. 1238) and allowed ranges within each Qos.

6.49.2 Field Documentation

6.49.2.1 entity_factory

```
struct DDS_EntityFactoryQosPolicy DDS_DomainParticipantFactoryQos::entity_factory
```

Entity factory policy, **ENTITY_FACTORY** (p. 1289).

6.49.2.2 resource_limits

```
struct DDS_SystemResourceLimitsQosPolicy DDS_DomainParticipantFactoryQos::resource_limits
```

<<*extension*>> (p. 1012) System resource limits, **SYSTEM_RESOURCE_LIMITS** (p. 1328).

Note: This QoS policy cannot be configured from XML configuration files.

6.49.2.3 profile

```
struct DDS_ProfileQosPolicy DDS_DomainParticipantFactoryQos::profile
```

<<*extension*>> (p. 1012) Qos profile policy, **PROFILE** (p. 1305).

Note: This QoS policy cannot be configured from XML configuration files.

6.49.2.4 logging

```
struct DDS_LoggingQosPolicy DDS_DomainParticipantFactoryQos::logging
```

<<*extension*>> (p. 1012) Logging qos policy, **LOGGING** (p. 1298).

6.49.2.5 monitoring

```
struct DDS_MonitoringQosPolicy DDS_DomainParticipantFactoryQos::monitoring
```

<<*extension*>> (p. 1012) Monitoring qos policy, **MONITORING** (p. 1299).

This QoS policy is used to collect and emit telemetry data of a Connex application using RTI Monitoring Library 2.0.

6.50 DDS_DomainParticipantListener Struct Reference

<<*interface*>> (p. 1013) Listener for participant status.

Data Fields

- struct **DDS_Listener as_listener**
A superclass instance of this **DDS_DomainParticipantListener** (p. 1723).
- struct **DDS_TopicListener as_topiclistener**
A superclass instance of this **DDS_DomainParticipantListener** (p. 1723).
- struct **DDS_PublisherListener as_publisherlistener**
A superclass instance of this **DDS_DomainParticipantListener** (p. 1723).
- struct **DDS_SubscriberListener as_subscriberlistener**
A superclass instance of this **DDS_DomainParticipantListener** (p. 1723).
- **DDS_DomainParticipantListener_InvalidLocalAdvanceNoticeStatusCallback on_invalid_local_status_↔ advance_notice**
Notifies the user that one of the local **DDS_DomainParticipant** (p. 77) trusted artifacts (identity or permissions) is about to expire.

6.50.1 Detailed Description

<<*interface*>> (p. 1013) Listener for participant status.

Entity:

DDS_DomainParticipant (p. 77)

Status:

Status Kinds (p. 1222)

This is the interface that can be implemented by an application-provided class and then registered with the **DDS_↔DomainParticipant** (p. 77) such that the application can be notified by RTI Connext of relevant status changes.

The **DDS_DomainParticipantListener** (p. 1723) interface extends all other Listener interfaces and has no additional operation beyond the ones defined by the more general listeners.

The purpose of the **DDS_DomainParticipantListener** (p. 1723) is to be the listener of last resort that is notified of all status changes not captured by more specific listeners attached to the **DDS_DomainEntity** (p. 1363) objects. When a relevant status change occurs, RTI Connext will first attempt to notify the listener attached to the concerned **DDS_↔DomainEntity** (p. 1363) if one is installed. Otherwise, RTI Connext will notify the Listener attached to the **DDS_↔DomainParticipant** (p. 77).

Important: Because a **DDS_DomainParticipantListener** (p. 1723) 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 **DDS_DomainParticipantListener** (p. 1723) 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

DDS_Listener (p. 1816)

DDS_DomainParticipant_set_listener (p. 154)

6.50.2 Field Documentation

6.50.2.1 as_listener

```
struct DDS_Listener DDS_DomainParticipantListener::as_listener
```

A superclass instance of this **DDS_DomainParticipantListener** (p. 1723).

6.50.2.2 as_topiclistener

```
struct DDS_TopicListener DDS_DomainParticipantListener::as_topiclistener
```

A superclass instance of this **DDS_DomainParticipantListener** (p. 1723).

6.50.2.3 as_publisherlistener

```
struct DDS_PublisherListener DDS_DomainParticipantListener::as_publisherlistener
```

A superclass instance of this **DDS_DomainParticipantListener** (p. 1723).

6.50.2.4 as_subscriberlistener

```
struct DDS_SubscriberListener DDS_DomainParticipantListener::as_subscriberlistener
```

A superclass instance of this **DDS_DomainParticipantListener** (p. 1723).

6.50.2.5 on_invalid_local_status_advance_notice

```
DDS_DomainParticipantListener_InvalidLocalAdvanceNoticeStatusCallback DDS_DomainParticipant↔  
Listener::on_invalid_local_status_advance_notice
```

Notifies the user that one of the local **DDS_DomainParticipant** (p. 77) trusted artifacts (identity or permissions) is about to expire.

6.51 DDS_DomainParticipantProtocolStatus Struct Reference

<<**extension**>> (p. 1012) 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.

Data Fields

- **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.

6.51.1 Detailed Description

<<*extension*>> (p. 1012) 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:

DDS_DomainParticipant (p. 77) The corrupted messages are detected by validating the received CRC. The participant protocol status can be obtained by enabling **DDS_WireProtocolQosPolicy::compute_crc** (p. 2100) and **DDS_WireProtocolQosPolicy::check_crc** (p. 2100) at the publishing and subscribing application respectively.

6.51.2 Field Documentation

6.51.2.1 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.

6.51.2.2 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.

6.51.2.3 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.

6.52 DDS_DomainParticipantQos Struct Reference

QoS policies supported by a **DDS_DomainParticipant** (p. 77) entity.

Data Fields

- struct **DDS_UserDataQosPolicy** **user_data**
User data policy, **USER_DATA** (p. 1343).
- struct **DDS_EntityFactoryQosPolicy** **entity_factory**
<<**basic**>> (p. 783) Entity factory policy, **ENTITY_FACTORY** (p. 1289).
- struct **DDS_WireProtocolQosPolicy** **wire_protocol**
<<**extension**>> (p. 1012) Wire Protocol policy, **WIRE_PROTOCOL** (p. 1344).
- struct **DDS_TransportBuiltinQosPolicy** **transport_builtin**
<<**extension**>> (p. 1012) Transport Builtin policy, **TRANSPORT_BUILTIN** (p. 1331).
- struct **DDS_TransportUnicastQosPolicy** **default_unicast**
<<**extension**>> (p. 1012) Default Unicast Transport policy, **TRANSPORT_UNICAST** (p. 1339).
- struct **DDS_DiscoveryQosPolicy** **discovery**
<<**extension**>> (p. 1012) Discovery policy, **DISCOVERY** (p. 1275).
- struct **DDS_DomainParticipantResourceLimitsQosPolicy** **resource_limits**
<<**extension**>> (p. 1012) Domain participant resource limits policy, **DOMAIN_PARTICIPANT_RESOURCE_LIMITS** (p. 1283).
- struct **DDS_EventQosPolicy** **event**
<<**extension**>> (p. 1012) Event policy, **EVENT** (p. 1291).
- struct **DDS_ReceiverPoolQosPolicy** **receiver_pool**
<<**extension**>> (p. 1012) Receiver pool policy, **RECEIVER_POOL** (p. 1320).
- struct **DDS_DatabaseQosPolicy** **database**
<<**extension**>> (p. 1012) Database policy, **DATABASE** (p. 1252).
- struct **DDS_DiscoveryConfigQosPolicy** **discovery_config**
<<**extension**>> (p. 1012) Discovery config policy, **DISCOVERY_CONFIG** (p. 1276).
- struct **DDS_PropertyQosPolicy** **property**
<<**basic**>> (p. 783) <<**extension**>> (p. 1012) Property policy, **PROPERTY** (p. 1306). See also *Property Reference Guide*.
- struct **DDS_EntityNameQosPolicy** **participant_name**
<<**basic**>> (p. 783) <<**extension**>> (p. 1012) The participant name. **ENTITY_NAME** (p. 1290)
- struct **DDS_TransportMulticastMappingQosPolicy** **multicast_mapping**
<<**extension**>> (p. 1012) The multicast mapping policy. **TRANSPORT_MULTICAST_MAPPING** (p. 1336)
- struct **DDS_ServiceQosPolicy** **service**
<<**extension**>> (p. 1012) The service qos policy. **SERVICE** (p. 1326)
- struct **DDS_PartitionQosPolicy** **partition**
<<**basic**>> (p. 783) <<**extension**>> (p. 1012) The partition qos policy. **PARTITION** (p. 1303)
- struct **DDS_TypeSupportQosPolicy** **type_support**
<<**extension**>> (p. 1012) Type support data, **TYPESUPPORT** (p. 1341).

6.52.1 Detailed Description

QoS policies supported by a **DDS_DomainParticipant** (p. 77) entity.

Certain members must be set in a consistent manner:

Length of **DDS_DomainParticipantQos::user_data** (p. 1728) .value <= **DDS_DomainParticipantQos::resource_limits** (p. 1729) .participant_user_data_max_length

For **DDS_DomainParticipantQos::discovery_config** (p. 1730) .publication_writer
 high_watermark <= **DDS_DomainParticipantQos::resource_limits** (p. 1729) .local_writer_allocation .max_↵
 count heartbeats_per_max_samples <= **DDS_DomainParticipantQos::resource_limits** (p. 1729) .local_writer_↵
 allocation.max_count

For **DDS_DomainParticipantQos::discovery_config** (p. 1730) .subscription_writer
 high_watermark <= **DDS_DomainParticipantQos::resource_limits** (p. 1729) .local_reader_allocation.max_↵
 count heartbeats_per_max_samples <= **DDS_DomainParticipantQos::resource_limits** (p. 1729) .local_reader_↵
 _allocation.max_count

If any of the above are not true, **DDS_DomainParticipant_set_qos** (p. 151) and **DDS_DomainParticipant_set_qos_↵
 _with_profile** (p. 152) and **DDS_DomainParticipantFactory_set_default_participant_qos** (p. 39) will fail with **DDS_↵
 _RETCODE_INCONSISTENT_POLICY** (p. 1222), and **DDS_DomainParticipantFactory_create_participant** (p. 41)
 will fail.

Entity:

DDS_DomainParticipant (p. 77)

See also

All QoS Policies (p. 1238) and allowed ranges within each Qos.

NDDS_DISCOVERY_PEERS (p. 1352)

6.52.2 Field Documentation

6.52.2.1 user_data

```
struct DDS_UserDataQosPolicy DDS_DomainParticipantQos::user_data
```

User data policy, **USER_DATA** (p. 1343).

6.52.2.2 entity_factory

```
struct DDS_EntityFactoryQosPolicy DDS_DomainParticipantQos::entity_factory
```

<<**basic**>> (p. 783) Entity factory policy, **ENTITY_FACTORY** (p. 1289).

6.52.2.3 wire_protocol

```
struct DDS_WireProtocolQosPolicy DDS_DomainParticipantQos::wire_protocol
```

<<*extension*>> (p. 1012) Wire Protocol policy, **WIRE_PROTOCOL** (p. 1344).

The wire protocol (RTPS) attributes associated with the participant.

6.52.2.4 transport_builtin

```
struct DDS_TransportBuiltinQosPolicy DDS_DomainParticipantQos::transport_builtin
```

<<*extension*>> (p. 1012) Transport Builtin policy, **TRANSPORT_BUILTIN** (p. 1331).

6.52.2.5 default_unicast

```
struct DDS_TransportUnicastQosPolicy DDS_DomainParticipantQos::default_unicast
```

<<*extension*>> (p. 1012) Default Unicast Transport policy, **TRANSPORT_UNICAST** (p. 1339).

6.52.2.6 discovery

```
struct DDS_DiscoveryQosPolicy DDS_DomainParticipantQos::discovery
```

<<*extension*>> (p. 1012) Discovery policy, **DISCOVERY** (p. 1275).

6.52.2.7 resource_limits

```
struct DDS_DomainParticipantResourceLimitsQosPolicy DDS_DomainParticipantQos::resource_limits
```

<<*extension*>> (p. 1012) Domain participant resource limits policy, **DOMAIN_PARTICIPANT_RESOURCE_LIMITS** (p. 1283).

6.52.2.8 event

```
struct DDS_EventQosPolicy DDS_DomainParticipantQos::event
```

<<*extension*>> (p. 1012) Event policy, **EVENT** (p. 1291).

6.52.2.9 receiver_pool

```
struct DDS_ReceiverPoolQosPolicy DDS_DomainParticipantQos::receiver_pool
```

<<**extension**>> (p. 1012) Receiver pool policy, **RECEIVER_POOL** (p. 1320).

6.52.2.10 database

```
struct DDS_DatabaseQosPolicy DDS_DomainParticipantQos::database
```

<<**extension**>> (p. 1012) Database policy, **DATABASE** (p. 1252).

6.52.2.11 discovery_config

```
struct DDS_DiscoveryConfigQosPolicy DDS_DomainParticipantQos::discovery_config
```

<<**extension**>> (p. 1012) Discovery config policy, **DISCOVERY_CONFIG** (p. 1276).

6.52.2.12 property

```
struct DDS_PropertyQosPolicy DDS_DomainParticipantQos::property
```

<<**basic**>> (p. 783) <<**extension**>> (p. 1012) Property policy, **PROPERTY** (p. 1306). See also `Property Reference Guide`.

6.52.2.13 participant_name

```
struct DDS_EntityNameQosPolicy DDS_DomainParticipantQos::participant_name
```

<<**basic**>> (p. 783) <<**extension**>> (p. 1012) The participant name. **ENTITY_NAME** (p. 1290)

6.52.2.14 multicast_mapping

```
struct DDS_TransportMulticastMappingQosPolicy DDS_DomainParticipantQos::multicast_mapping
```

<<**extension**>> (p. 1012) The multicast mapping policy. **TRANSPORT_MULTICAST_MAPPING** (p. 1336)

6.52.2.15 service

```
struct DDS_ServiceQosPolicy DDS_DomainParticipantQos::service
```

<<**extension**>> (p. 1012) The service qos policy. **SERVICE** (p. 1326)

6.52.2.16 partition

```
struct DDS_PartitionQosPolicy DDS_DomainParticipantQos::partition
```

<<**basic**>> (p. 783) <<**extension**>> (p. 1012) The partition qos policy. **PARTITION** (p. 1303)

6.52.2.17 type_support

```
struct DDS_TypeSupportQosPolicy DDS_DomainParticipantQos::type_support
```

<<**extension**>> (p. 1012) Type support data, **TYPESUPPORT** (p. 1341).

Optional value that is passed to a type plugin's on_participant_attached function.

6.53 DDS_DomainParticipantResourceLimitsQosPolicy Struct Reference

<<**extension**>> (p. 795) Various settings that configure how a **DDS_DomainParticipant** (p. 77) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.

Data Fields

- struct **DDS_AllocationSettings_t local_writer_allocation**
 <<**extension**>> (p. 795) Allocation settings applied to local DataWriters.
- struct **DDS_AllocationSettings_t local_reader_allocation**
 <<**extension**>> (p. 795) Allocation settings applied to local DataReaders.
- struct **DDS_AllocationSettings_t local_publisher_allocation**
 <<**extension**>> (p. 795) Allocation settings applied to local Publisher.
- struct **DDS_AllocationSettings_t local_subscriber_allocation**
 <<**extension**>> (p. 795) Allocation settings applied to local Subscriber.
- struct **DDS_AllocationSettings_t local_topic_allocation**
 <<**extension**>> (p. 795) Allocation settings applied to local Topic.
- struct **DDS_AllocationSettings_t remote_writer_allocation**
 <<**extension**>> (p. 795) Allocation settings applied to remote DataWriters.
- struct **DDS_AllocationSettings_t remote_reader_allocation**

- <<*extension*>> (p. 795) Allocation settings applied to remote DataReaders.
- struct **DDS_AllocationSettings_t remote_participant_allocation**
 - <<*extension*>> (p. 795) Allocation settings applied to remote DomainParticipants.
- struct **DDS_AllocationSettings_t matching_writer_reader_pair_allocation**
 - <<*extension*>> (p. 795) Allocation settings applied to matching local writer and remote/local reader pairs.
- struct **DDS_AllocationSettings_t matching_reader_writer_pair_allocation**
 - <<*extension*>> (p. 795) Allocation settings applied to matching local reader and remote/local writer pairs.
- struct **DDS_AllocationSettings_t ignored_entity_allocation**
 - <<*extension*>> (p. 795) Allocation settings applied to ignored entities.
- struct **DDS_AllocationSettings_t content_filtered_topic_allocation**
 - <<*extension*>> (p. 795) Allocation settings applied to content filtered topic.
- struct **DDS_AllocationSettings_t content_filter_allocation**
 - <<*extension*>> (p. 795) Allocation settings applied to content filter.
- struct **DDS_AllocationSettings_t read_condition_allocation**
 - <<*extension*>> (p. 795) Allocation settings applied to read condition pool.
- struct **DDS_AllocationSettings_t query_condition_allocation**
 - <<*extension*>> (p. 795) Allocation settings applied to query condition pool.
- struct **DDS_AllocationSettings_t outstanding_asynchronous_sample_allocation**
 - <<*extension*>> (p. 795) Allocation settings applied to the maximum number of samples (from all **DDS_DataWriter** (p. 485)) waiting to be asynchronously written.
- struct **DDS_AllocationSettings_t flow_controller_allocation**
 - <<*extension*>> (p. 795) Allocation settings applied to flow controllers.
- **DDS_Long local_writer_hash_buckets**
 - [DEPRECATED] (p. 965) <<*extension*>> (p. 795) Hash_Buckets settings applied to local DataWriters.
- **DDS_Long local_reader_hash_buckets**
 - [DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for local DataReaders.
- **DDS_Long local_publisher_hash_buckets**
 - [DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for local Publisher.
- **DDS_Long local_subscriber_hash_buckets**
 - [DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for local Subscriber.
- **DDS_Long local_topic_hash_buckets**
 - [DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for local Topic.
- **DDS_Long remote_writer_hash_buckets**
 - [DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for remote DataWriters.
- **DDS_Long remote_reader_hash_buckets**
 - [DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for remote DataReaders.
- **DDS_Long remote_participant_hash_buckets**
 - [DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for remote DomainParticipants.
- **DDS_Long matching_writer_reader_pair_hash_buckets**
 - [DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for matching local writer and remote/local reader pairs.
- **DDS_Long matching_reader_writer_pair_hash_buckets**
 - [DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for matching local reader and remote/local writer pairs.
- **DDS_Long ignored_entity_hash_buckets**
 - [DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for ignored entities.
- **DDS_Long content_filtered_topic_hash_buckets**
 - [DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for content filtered topics.

- **DDS_Long content_filter_hash_buckets**
[DEPRECATED] (p. 965) <<extension>> (p. 795) Number of hash buckets for content filters.
- **DDS_Long flow_controller_hash_buckets**
[DEPRECATED] (p. 965) <<extension>> (p. 795) Number of hash buckets for flow controllers.
- **DDS_Long max_gather_destinations**
<<extension>> (p. 795) Maximum number of destinations per RTI Connex send.
- **DDS_Long participant_user_data_max_length**
<<extension>> (p. 795) Maximum length of user data in **DDS_DomainParticipantQos** (p. 1726) and **DDS_ParticipantBuiltinTopicData** (p. 1870)
- **DDS_Long topic_data_max_length**
<<extension>> (p. 795) Maximum length of topic data in **DDS_TopicQos** (p. 2042), **DDS_TopicBuiltinTopicData** (p. 2034), **DDS_PublicationBuiltinTopicData** (p. 1905) and **DDS_SubscriptionBuiltinTopicData** (p. 2012)
- **DDS_Long publisher_group_data_max_length**
<<extension>> (p. 795) Maximum length of group data in **DDS_PublisherQos** (p. 1918) and **DDS_PublicationBuiltinTopicData** (p. 1905)
- **DDS_Long subscriber_group_data_max_length**
<<extension>> (p. 795) Maximum length of group data in **DDS_SubscriberQos** (p. 2010) and **DDS_SubscriptionBuiltinTopicData** (p. 2012)
- **DDS_Long writer_user_data_max_length**
<<extension>> (p. 795) Maximum length of user data in **DDS_DataWriterQos** (p. 1666) and **DDS_PublicationBuiltinTopicData** (p. 1905)
- **DDS_Long reader_user_data_max_length**
<<extension>> (p. 795) Maximum length of user data in **DDS_DataReaderQos** (p. 1611) and **DDS_SubscriptionBuiltinTopicData** (p. 2012)
- **DDS_Long max_partitions**
<<extension>> (p. 795) Maximum number of partition name strings allowable in a **DDS_PartitionQosPolicy** (p. 1881).
- **DDS_Long max_partition_cumulative_characters**
<<extension>> (p. 795) Maximum number of combined characters allowable in all partition names in a **DDS_PartitionQosPolicy** (p. 1881).
- **DDS_Long type_code_max_serialized_length**
[DEPRECATED] (p. 965) <<extension>> (p. 795) Maximum size of serialized string for type code.
- **DDS_Long type_object_max_serialized_length**
<<extension>> (p. 795) The maximum length, in bytes, that the buffer to serialize a **TypeObject** can consume.
- **DDS_Long serialized_type_object_dynamic_allocation_threshold**
<<extension>> (p. 795) A threshold, in bytes, for dynamic memory allocation for the serialized **TypeObject**.
- **DDS_Long type_object_max_deserialized_length**
<<extension>> (p. 795) The maximum number of bytes that a deserialized **TypeObject** can consume.
- **DDS_Long deserialized_type_object_dynamic_allocation_threshold**
<<extension>> (p. 795) A threshold, in bytes, for dynamic memory allocation for the deserialized **TypeObject**.
- **DDS_Long contentfilter_property_max_length**
<<extension>> (p. 795) This field is the maximum length of all data related to a Content-filtered topic.
- **DDS_Long channel_seq_max_length**
<<extension>> (p. 795) Maximum number of channels that can be specified in **DDS_MultiChannelQosPolicy** (p. 1856) for **MultiChannel DataWriters**.
- **DDS_Long channel_filter_expression_max_length**
<<extension>> (p. 795) Maximum length of a channel **DDS_ChannelSettings_t::filter_expression** (p. 1564) in a **MultiChannel DataWriter**.
- **DDS_Long participant_property_list_max_length**
<<extension>> (p. 795) Maximum number of properties associated with the **DDS_DomainParticipant** (p. 77).

- **DDS_Long participant_property_string_max_length**
 <<extension>> (p. 795) Maximum string length of the properties associated with the **DDS_DomainParticipant** (p. 77).
- **DDS_Long writer_property_list_max_length**
 <<extension>> (p. 795) Maximum number of properties associated with a **DDS_DataWriter** (p. 485).
- **DDS_Long writer_property_string_max_length**
 <<extension>> (p. 795) Maximum string length of the properties associated with a **DDS_DataWriter** (p. 485).
- **DDS_Long reader_property_list_max_length**
 <<extension>> (p. 795) Maximum number of properties associated with a **DDS_DataReader** (p. 615).
- **DDS_Long reader_property_string_max_length**
 <<extension>> (p. 795) Maximum string length of the properties associated with a **DDS_DataReader** (p. 615).
- **DDS_Long max_endpoint_groups**
 <<extension>> (p. 795) Maximum number of **DDS_EndpointGroup_t** (p. 1784) allowable in a **DDS_AvailabilityQosPolicy** (p. 1547).
- **DDS_Long max_endpoint_group_cumulative_characters**
 <<extension>> (p. 795) Maximum number of combined **role_name** characters allowed in all **DDS_EndpointGroup_t** (p. 1784) in a **DDS_AvailabilityQosPolicy** (p. 1547).
- **DDS_Long transport_info_list_max_length**
 <<extension>> (p. 795) Maximum number of installed transports to send and receive information about in **DDS_ParticipantBuiltinTopicData::transport_info** (p. 1873).
- **DDS_DomainParticipantResourceLimitsIgnoredEntityReplacementKind ignored_entity_replacement_kind**
 [DEPRECATED] (p. 965) <<extension>> (p. 795) Replacement policy for the ignored entities. It sets what entity can be replaced when resource limits set in **DDS_DomainParticipantResourceLimitsQosPolicy::ignored_entity_allocation** (p. 1738) are reached.
- struct **DDS_AllocationSettings_t remote_topic_query_allocation**
 <<extension>> (p. 795) Allocation settings applied to remote **TopicQueries**.
- **DDS_Long remote_topic_query_hash_buckets**
 [DEPRECATED] (p. 965) <<extension>> (p. 795) Number of hash buckets for remote **TopicQueries**.
- **DDS_Long writer_data_tag_list_max_length**
 <<extension>> (p. 795) Maximum number of data tags associated with a **DDS_DataWriter** (p. 485).
- **DDS_Long writer_data_tag_string_max_length**
 <<extension>> (p. 795) Maximum string length of the data tags associated with a **DDS_DataWriter** (p. 485).
- **DDS_Long reader_data_tag_list_max_length**
 <<extension>> (p. 795) Maximum number of data tags associated with a **DDS_DataReader** (p. 615).
- **DDS_Long reader_data_tag_string_max_length**
 <<extension>> (p. 795) Maximum string length of the data tags associated with a **DDS_DataReader** (p. 615).
- **DDS_UnsignedLong shm_ref_transfer_mode_max_segments**
 <<extension>> (p. 795) Maximum number of segments created by all **DataWriters** belonging to a **DDS_DomainParticipant** (p. 77).

6.53.1 Detailed Description

<<extension>> (p. 795) Various settings that configure how a **DDS_DomainParticipant** (p. 77) 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 **DDS_DomainParticipant** (p. 77) is allowed to dynamically allocate memory as needed as users create local Entities such as **DDS_DataWriter** (p. 485) and **DDS_DataReader** (p. 615) or as the participant discovers new applications to store their information. By setting fixed values for the maximum parameters in this QoS policy, 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 **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615) 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. 1881), the maximum length of data store in the **DDS_UserDataQosPolicy** (p. 2086) and **DDS_GroupDataQosPolicy** (p. 1803), 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 **DDS_DomainParticipant** (p. 77) to another, a **DDS_DomainParticipant** (p. 77) 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_DomainParticipantResourceLimitsQosPolicy::type_object_max_serialized_length** (p. 1749).

This QoS policy is an extension to the DDS standard.

Entity:

DDS_DomainParticipant (p. 77)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

6.53.2 Field Documentation

6.53.2.1 local_writer_allocation

```
struct DDS_AllocationSettings_t DDS_DomainParticipantResourceLimitsQosPolicy::local_writer_↵
allocation
```

<<**extension**>> (p. 795) Allocation settings applied to local DataWriters.

[default] initial_count = 16; max_count = **DDS_LENGTH_UNLIMITED** (p. 1326); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 1537)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.2 local_reader_allocation

```
struct DDS_AllocationSettings_t DDS_DomainParticipantResourceLimitsQosPolicy::local_reader_↔
allocation
```

<<**extension**>> (p. 795) Allocation settings applied to local DataReaders.

[default] initial_count = 16; max_count = **DDS_LENGTH_UNLIMITED** (p. 1326); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 1537)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.3 local_publisher_allocation

```
struct DDS_AllocationSettings_t DDS_DomainParticipantResourceLimitsQosPolicy::local_publisher_↔
allocation
```

<<**extension**>> (p. 795) Allocation settings applied to local Publisher.

[default] initial_count = 4; max_count = **DDS_LENGTH_UNLIMITED** (p. 1326); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 1537)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.4 local_subscriber_allocation

```
struct DDS_AllocationSettings_t DDS_DomainParticipantResourceLimitsQosPolicy::local_subscriber_↔
allocation
```

<<**extension**>> (p. 795) Allocation settings applied to local Subscriber.

[default] initial_count = 4; max_count = **DDS_LENGTH_UNLIMITED** (p. 1326); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 1537)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.5 local_topic_allocation

```
struct DDS_AllocationSettings_t DDS_DomainParticipantResourceLimitsQosPolicy::local_topic_↵  
allocation
```

<<**extension**>> (p. 795) Allocation settings applied to local Topic.

[default] initial_count = 16; max_count = **DDS_LENGTH_UNLIMITED** (p. 1326); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 1537)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.6 remote_writer_allocation

```
struct DDS_AllocationSettings_t DDS_DomainParticipantResourceLimitsQosPolicy::remote_writer_↵  
allocation
```

<<**extension**>> (p. 795) 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. 1326); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 1537)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.7 remote_reader_allocation

```
struct DDS_AllocationSettings_t DDS_DomainParticipantResourceLimitsQosPolicy::remote_reader_↵  
allocation
```

<<**extension**>> (p. 795) 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. 1326); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 1537)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.8 remote_participant_allocation

```
struct DDS_AllocationSettings_t DDS_DomainParticipantResourceLimitsQosPolicy::remote_participant↔
_allocation
```

<<**extension**>> (p. 795) 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. 1326); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 1537)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.9 matching_writer_reader_pair_allocation

```
struct DDS_AllocationSettings_t DDS_DomainParticipantResourceLimitsQosPolicy::matching_writer↔
_reader_pair_allocation
```

<<**extension**>> (p. 795) Allocation settings applied to matching local writer and remote/local reader pairs.

[default] initial_count = 32; max_count = **DDS_LENGTH_UNLIMITED** (p. 1326); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 1537)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.10 matching_reader_writer_pair_allocation

```
struct DDS_AllocationSettings_t DDS_DomainParticipantResourceLimitsQosPolicy::matching_reader↔
_writer_pair_allocation
```

<<**extension**>> (p. 795) Allocation settings applied to matching local reader and remote/local writer pairs.

[default] initial_count = 32; max_count = **DDS_LENGTH_UNLIMITED** (p. 1326); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 1537)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.11 ignored_entity_allocation

```
struct DDS_AllocationSettings_t DDS_DomainParticipantResourceLimitsQosPolicy::ignored_entity_↔
allocation
```

<<**extension**>> (p. 795) Allocation settings applied to ignored entities.

[default] initial_count = 8; max_count = **DDS_LENGTH_UNLIMITED** (p. 1326); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 1537)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.12 content_filtered_topic_allocation

```
struct DDS_AllocationSettings_t DDS_DomainParticipantResourceLimitsQosPolicy::content_filtered_↔
topic_allocation
```

<<**extension**>> (p. 795) Allocation settings applied to content filtered topic.

[default] initial_count = 4; max_count = **DDS_LENGTH_UNLIMITED** (p. 1326); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 1537)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Content Filtering** (p. 841)

6.53.2.13 content_filter_allocation

```
struct DDS_AllocationSettings_t DDS_DomainParticipantResourceLimitsQosPolicy::content_filter_↔
allocation
```

<<**extension**>> (p. 795) Allocation settings applied to content filter.

[default] initial_count = 4; max_count = **DDS_LENGTH_UNLIMITED** (p. 1326); incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 1537)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Content Filtering** (p. 841)

6.53.2.14 read_condition_allocation

```
struct DDS_AllocationSettings_t DDS_DomainParticipantResourceLimitsQosPolicy::read_condition_↔
allocation
```

<<**extension**>> (p. 795) Allocation settings applied to read condition pool.

[default] initial_count = 4; max_count = **DDS_LENGTH_UNLIMITED** (p. 1326), incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 1537)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.15 query_condition_allocation

```
struct DDS_AllocationSettings_t DDS_DomainParticipantResourceLimitsQosPolicy::query_condition_↔
allocation
```

<<**extension**>> (p. 795) Allocation settings applied to query condition pool.

[default] initial_count = 4; max_count = **DDS_LENGTH_UNLIMITED** (p. 1326), incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 1537)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Content Filtering** (p. 841)

6.53.2.16 outstanding_asynchronous_sample_allocation

```
struct DDS_AllocationSettings_t DDS_DomainParticipantResourceLimitsQosPolicy::outstanding_↔
asynchronous_sample_allocation
```

<<**extension**>> (p. 795) Allocation settings applied to the maximum number of samples (from all **DDS_DataWriter** (p. 485)) waiting to be asynchronously written.

[default] initial_count = 64; max_count = **DDS_LENGTH_UNLIMITED** (p. 1326), incremental_count = -1

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 1537)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Flow Control** (p. 863)

6.53.2.17 flow_controller_allocation

```
struct DDS_AllocationSettings_t DDS_DomainParticipantResourceLimitsQosPolicy::flow_controller_allocation
```

<<**extension**>> (p. 795) Allocation settings applied to flow controllers.

[**default**] initial_count = 4; max_count = **DDS_LENGTH_UNLIMITED** (p. 1326), incremental_count = -1

[**range**] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 1537)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Flow Control** (p. 863)

6.53.2.18 local_writer_hash_buckets

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::local_writer_hash_buckets
```

[**DEPRECATED**] (p. 965) <<**extension**>> (p. 795) Hash_Buckets settings applied to local DataWriters.

[**default**] 4

[**range**] [1, 10000]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.19 local_reader_hash_buckets

```
DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::local_reader_hash_buckets
```

[**DEPRECATED**] (p. 965) <<**extension**>> (p. 795) Number of hash buckets for local DataReaders.

[**default**] 4

[**range**] [1, 10000]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.20 local_publisher_hash_buckets

`DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::local_publisher_hash_buckets`

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for local Publisher.

[default] 1

[range] [1, 10000]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.21 local_subscriber_hash_buckets

`DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::local_subscriber_hash_buckets`

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for local Subscriber.

[default] 1

[range] [1, 10000]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.22 local_topic_hash_buckets

`DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::local_topic_hash_buckets`

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for local Topic.

[default] 4

[range] [1, 10000]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.23 remote_writer_hash_buckets

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::remote_writer_hash_buckets

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for remote DataWriters.

Remote DataWriters include all DataWriters, both local and remote.

[default] 16

[range] [1, 10000]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.24 remote_reader_hash_buckets

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::remote_reader_hash_buckets

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for remote DataReaders.

Remote DataReaders include all DataReaders, both local and remote.

[default] 16

[range] [1, 10000]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.25 remote_participant_hash_buckets

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::remote_participant_hash_buckets

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for remote DomainParticipants.

Remote DomainParticipants include all DomainParticipants, both local and remote.

[default] 4

[range] [1, 10000]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.26 matching_writer_reader_pair_hash_buckets

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::matching_writer_reader_pair_hash_buckets

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for matching local writer and remote/local reader pairs.

[default] 32

[range] [1, 10000]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.27 matching_reader_writer_pair_hash_buckets

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::matching_reader_writer_pair_hash_buckets

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for matching local reader and remote/local writer pairs.

[default] 32

[range] [1, 10000]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.28 ignored_entity_hash_buckets

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::ignored_entity_hash_buckets

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for ignored entities.

[default] 1

[range] [1, 10000]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.29 content_filtered_topic_hash_buckets

`DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::content_filtered_topic_hash_buckets`

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for content filtered topics.

[default] 1

[range] [1, 10000]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Content Filtering** (p. 841)

6.53.2.30 content_filter_hash_buckets

`DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::content_filter_hash_buckets`

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for content filters.

[default] 1

[range] [1, 10000]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Content Filtering** (p. 841)

6.53.2.31 flow_controller_hash_buckets

`DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::flow_controller_hash_buckets`

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for flow controllers.

[default] 1

[range] [1, 10000]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Flow Control** (p. 863)

6.53.2.32 max_gather_destinations

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::max_gather_destinations

<<**extension**>> (p. 795) Maximum number of destinations per RTI Connex send.

When RTI Connex sends out a message, it has the capability to send to multiple destinations to be more efficient. The maximum number of destinations per RTI Connex send is specified by max_gather_destinations.

[default] 16

[range] [16, 1 million]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.33 participant_user_data_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::participant_user_data_max_length

<<**extension**>> (p. 795) Maximum length of user data in **DDS_DomainParticipantQos** (p. 1726) and **DDS_ParticipantBuiltinTopicData** (p. 1870)

[default] 256

[range] [0,0x7fffffff]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Application-Defined Metadata** (p. 836)

6.53.2.34 topic_data_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::topic_data_max_length

<<**extension**>> (p. 795) Maximum length of topic data in **DDS_TopicQos** (p. 2042), **DDS_TopicBuiltinTopicData** (p. 2034), **DDS_PublicationBuiltinTopicData** (p. 1905) and **DDS_SubscriptionBuiltinTopicData** (p. 2012)

[default] 256

[range] [0,0x7fffffff]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Application-Defined Metadata** (p. 836)

6.53.2.35 publisher_group_data_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::publisher_group_data_max_length

<<*extension*>> (p. 795) Maximum length of group data in **DDS_PublisherQos** (p. 1918) and **DDS_Publication**↔
BuiltinTopicData (p. 1905)

[default] 256

[range] [0,0x7ffffff]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Application-Defined Metadata** (p. 836)

6.53.2.36 subscriber_group_data_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::subscriber_group_data_max_length

<<*extension*>> (p. 795) Maximum length of group data in **DDS_SubscriberQos** (p. 2010) and **DDS_Subscription**↔
BuiltinTopicData (p. 2012)

[default] 256

[range] [0,0x7ffffff]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Application-Defined Metadata** (p. 836)

6.53.2.37 writer_user_data_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::writer_user_data_max_length

<<*extension*>> (p. 795) Maximum length of user data in **DDS_DataWriterQos** (p. 1666) and **DDS_Publication**↔
BuiltinTopicData (p. 1905)

[default] 256

[range] [0,0x7ffffff]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Application-Defined Metadata** (p. 836)

6.53.2.38 reader_user_data_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::reader_user_data_max_length

<<**extension**>> (p. 795) Maximum length of user data in **DDS_DataReaderQos** (p. 1611) and **DDS_SubscriptionBuiltInTopicData** (p. 2012)

[default] 256

[range] [0,0x7fffffff]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Application-Defined Metadata** (p. 836)

6.53.2.39 max_partitions

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::max_partitions

<<**extension**>> (p. 795) Maximum number of partition name strings allowable in a **DDS_PartitionQosPolicy** (p. 1881).

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. 914), **Middleware Memory Management** (p. 878), **Discovery Isolation** (p. 858)

6.53.2.40 max_partition_cumulative_characters

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::max_partition_cumulative_characters

<<**extension**>> (p. 795) Maximum number of combined characters allowable in all partition names in a **DDS_PartitionQosPolicy** (p. 1881).

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. 914), **Middleware Memory Management** (p. 878), **Discovery Isolation** (p. 858)

6.53.2.41 type_code_max_serialized_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::type_code_max_serialized_length

[DEPRECATED] (p. 965) <<**extension**>> (p. 795) Maximum size of serialized string for type code.

This parameter is an alternative to **DDS_DomainParticipantResourceLimitsQosPolicy::type_object_max_serialized_length** (p. 1749) for limiting the size of the type code that a **DDS_DomainParticipant** (p. 77) 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 is now the standard method of exchanging type information in RTI Connext. It is recommended to use **DDS_DomainParticipantResourceLimitsQosPolicy::type_object_max_serialized_length** (p. 1749) to configure the maximum serialized type object string.

[default] 0

[range] [0,0xffff]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Type System** (p. 912)

6.53.2.42 type_object_max_serialized_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::type_object_max_serialized_length

<<**extension**>> (p. 795) The maximum length, in bytes, that the buffer to serialize a TypeObject can consume.

This parameter limits the size of the TypeObject that a DomainParticipant is able to propagate. Since TypeObjects 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 TypeObjects larger than the default maximum of 8192 bytes. This field allows you to specify a larger value. The desired size for a given **DDS_TypeCode** (p.2071) can be obtained using **DDS_TypeCode_get_type_object_serialized_size** (p.277).

[default] 8192

[range] [0,0x7fffffff]

Categories

Immutable (p.914), **Middleware Memory Management** (p.878), **Type System** (p.912)

6.53.2.43 serialized_type_object_dynamic_allocation_threshold

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::serialized_type_object_dynamic_allocation↔_threshold

<<**extension**>> (p. 795) A threshold, in bytes, for dynamic memory allocation for the serialized TypeObject.

Above this threshold, the memory for a TypeObject 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_serialized_length** (p. 1749) is different than **DDS_LENGTH_UNLIMITED** (p.1326) and is smaller than **DDS_DomainParticipantResourceLimitsQos↔Policy::serialized_type_object_dynamic_allocation_threshold** (p.1750): **DDS_DomainParticipantResource↔LimitsQosPolicy::serialized_type_object_dynamic_allocation_threshold** (p.1750) will be adjusted to **DDS_↔DomainParticipantResourceLimitsQosPolicy::type_object_max_serialized_length** (p. 1749).

[default] 8192

[range] [0,0x7fffffff] ≤ **DDS_DomainParticipantResourceLimitsQosPolicy::type_object_max_serialized_length** (p. 1749)

Categories

Immutable (p.914), **Middleware Memory Management** (p.878), **Type System** (p.912)

6.53.2.44 type_object_max_deserialized_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::type_object_max_deserialized_length

<<**extension**>> (p. 795) The maximum number of bytes that a deserialized TypeObject can consume.

This parameter limits the size of the TypeObject that a DomainParticipant is able to store.

[default] DDS_LENGTH_UNLIMITED (p. 1326)

[range] [0,0x7fffffff] or DDS_LENGTH_UNLIMITED (p. 1326)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Type System** (p. 912)

6.53.2.45 deserialized_type_object_dynamic_allocation_threshold

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::deserialized_type_object_dynamic_allocation_threshold

<<**extension**>> (p. 795) A threshold, in bytes, for dynamic memory allocation for the deserialized TypeObject.

Above this threshold, the memory for a TypeObject 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. 1750) is different than **DDS_LENGTH_UNLIMITED** (p. 1326) and is smaller than **DDS_DomainParticipantResourceLimitsQosPolicy::deserialized_type_object_dynamic_allocation_threshold** (p. 1751): **DDS_DomainParticipantResourceLimitsQosPolicy::deserialized_type_object_dynamic_allocation_threshold** (p. 1751) will be adjusted to **DDS_DomainParticipantResourceLimitsQosPolicy::type_object_max_deserialized_length** (p. 1750).

[default] 4096

[range] [0,0x7fffffff] <= **DDS_DomainParticipantResourceLimitsQosPolicy::type_object_max_deserialized_length** (p. 1750)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Type System** (p. 912)

6.53.2.46 contentfilter_property_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::contentfilter_property_max_length

<<**extension**>> (p. 795) 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. 914), **Middleware Memory Management** (p. 878), **Content Filtering** (p. 841)

6.53.2.47 channel_seq_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::channel_seq_max_length

<<**extension**>> (p. 795) Maximum number of channels that can be specified in **DDS_MultiChannelQosPolicy** (p. 1856) for MultiChannel DataWriters.

[default] 32

[range] [0,0xffff]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Content Filtering** (p. 841)

6.53.2.48 channel_filter_expression_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::channel_filter_expression_max_length

<<**extension**>> (p. 795) Maximum length of a channel **DDS_ChannelSettings_t::filter_expression** (p. 1564) in a MultiChannel DataWriter.

The length should account for a terminating NULL ('\0') character.

[default] 256

[range] [0,0xffff]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Content Filtering** (p. 841)

6.53.2.49 participant_property_list_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::participant_property_list_max_length

<<**extension**>> (p. 795) Maximum number of properties associated with the **DDS_DomainParticipant** (p. 77).

[default] 32

[range] [0,0xffff]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Application-Defined Metadata** (p. 836)

6.53.2.50 participant_property_string_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::participant_property_string_max_length

<<**extension**>> (p. 795) Maximum string length of the properties associated with the **DDS_DomainParticipant** (p. 77).

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 **DDS_DomainParticipant** (p. 77) properties.

[default] 4096

[range] [0,0xffff]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Application-Defined Metadata** (p. 836)

6.53.2.51 writer_property_list_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::writer_property_list_max_length

<<**extension**>> (p. 795) Maximum number of properties associated with a **DDS_DataWriter** (p. 485).

[range] [0,0xffff]

[default] 32

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Application-Defined Metadata** (p. 836)

6.53.2.52 writer_property_string_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::writer_property_string_max_length

<<**extension**>> (p. 795) Maximum string length of the properties associated with a **DDS_DataWriter** (p. 485).

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 **DDS_DataWriter** (p. 485) properties.

[default] 1024

[range] [0,0xffff]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Application-Defined Metadata** (p. 836)

6.53.2.53 reader_property_list_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::reader_property_list_max_length

<<**extension**>> (p. 795) Maximum number of properties associated with a **DDS_DataReader** (p. 615).

[default] 32

[range] [0,0xffff]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Application-Defined Metadata** (p. 836)

6.53.2.54 reader_property_string_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::reader_property_string_max_length

<<**extension**>> (p. 795) Maximum string length of the properties associated with a **DDS_DataReader** (p. 615).

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 **DDS_DataReader** (p. 615) properties.

[default] 1024

[range] [0,0xffff]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Application-Defined Metadata** (p. 836)

6.53.2.55 max_endpoint_groups

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::max_endpoint_groups

<<**extension**>> (p. 795) Maximum number of **DDS_EndpointGroup_t** (p. 1784) allowable in a **DDS_AvailabilityQosPolicy** (p. 1547).

[default] 32

[range] [0,65535]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.56 max_endpoint_group_cumulative_characters

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::max_endpoint_group_cumulative_characters

<<**extension**>> (p. 795) Maximum number of combined role_name characters allowed in all **DDS_EndpointGroup_t** (p. 1784) in a **DDS_AvailabilityQosPolicy** (p. 1547).

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. 914), **Middleware Memory Management** (p. 878)

6.53.2.57 transport_info_list_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::transport_info_list_max_length

<<**extension**>> (p. 795) Maximum number of installed transports to send and receive information about in **DDS_ParticipantBuiltinTopicData::transport_info** (p. 1873).

[default] 12

[range] [0,100]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Transports** (p. 908)

6.53.2.58 ignored_entity_replacement_kind

DDS_DomainParticipantResourceLimitsIgnoredEntityReplacementKind DDS_DomainParticipantResourceLimitsQosPolicy::ignored_entity_replacement_kind

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Replacement policy for the ignored entities. It sets what entity can be replaced when resource limits set in **DDS_DomainParticipantResourceLimitsQosPolicy::ignored_entity_allocation** (p. 1738) are reached.

When a **DDS_DomainParticipant** (p. 77)'s number of ignored entities is greater than **DDS_DomainParticipantResourceLimitsQosPolicy::ignored_entity_allocation** (p. 1738), the **DDS_DomainParticipant** (p. 77) 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. 1284)

See also

DDS_DomainParticipantResourceLimitsIgnoredEntityReplacementKind (p. 1284)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.59 remote_topic_query_allocation

struct DDS_AllocationSettings_t DDS_DomainParticipantResourceLimitsQosPolicy::remote_topic_query_allocation

<<*extension*>> (p. 795) Allocation settings applied to remote TopicQueries.

Settings applied to the allocation of information about **DDS_TopicQuery** (p. 707) 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. 1326); `incremental_count = -1`

[range] See allowed ranges in struct **DDS_AllocationSettings_t** (p. 1537)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Topic Queries** (p. 911)

6.53.2.60 remote_topic_query_hash_buckets

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::remote_topic_query_hash_buckets

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Number of hash buckets for remote TopicQueries.

[default] 1

[range] [1, 10000]

See also

DDS_DomainParticipantResourceLimitsQosPolicy::remote_topic_query_allocation (p. 1756)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Topic Queries** (p. 911)

6.53.2.61 writer_data_tag_list_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::writer_data_tag_list_max_length

<<*extension*>> (p. 795) Maximum number of data tags associated with a **DDS_DataWriter** (p. 485).

[default] 0

[range] [0,0xffff]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.62 writer_data_tag_string_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::writer_data_tag_string_max_length

<<*extension*>> (p. 795) Maximum string length of the data tags associated with a **DDS_DataWriter** (p. 485).

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 **DDS_DataWriter** (p. 485) data tags.

[default] 0

[range] [0,0xffff]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.63 reader_data_tag_list_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::reader_data_tag_list_max_length

<<**extension**>> (p. 795) Maximum number of data tags associated with a **DDS_DataReader** (p. 615).

[default] 0

[range] [0,0xffff]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.64 reader_data_tag_string_max_length

DDS_Long DDS_DomainParticipantResourceLimitsQosPolicy::reader_data_tag_string_max_length

<<**extension**>> (p. 795) Maximum string length of the data tags associated with a **DDS_DataReader** (p. 615).

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 **DDS_DataReader** (p. 615) data tags.

[default] 0

[range] [0,0xffff]

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.53.2.65 shmem_ref_transfer_mode_max_segments

DDS_UnsignedLong DDS_DomainParticipantResourceLimitsQosPolicy::shmem_ref_transfer_mode_max_↵
segments

<<**extension**>> (p. 795) Maximum number of segments created by all DataWriters belonging to a **DDS_Domain**↵
Participant (p. 77).

[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 shmni.

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878), **Sending Large Data** (p. 871)

6.54 DDS_DomainParticipantSeq Struct Reference

Declares IDL sequence `< DDS_DomainParticipant (p. 77) >` .

6.54.1 Detailed Description

Declares IDL sequence `< DDS_DomainParticipant (p. 77) >` .

See also

FooSeq (p. 2113)

6.55 DDS_DoubleSeq Struct Reference

Instantiates **FooSeq** (p. 2113) `< DDS_Double (p. 1203) >`

6.55.1 Detailed Description

Instantiates **FooSeq** (p. 2113) `< DDS_Double (p. 1203) >`

Instantiates:

`<<generic>>` (p. 1013) **FooSeq** (p. 2113)

See also

DDS_Double (p. 1203)

FooSeq (p. 2113)

6.56 DDS_DurabilityQosPolicy Struct Reference

`<<spec>>` (p. 790) This QoS policy specifies whether or not RTI Connext will store and deliver previously published data samples to new **DDS_DataReader** (p. 615) entities that join the network later.

Data Fields

- **DDS_DurabilityQosPolicyKind kind**
`<<basic>>` (p. 783) `<<spec>>` (p. 790) *The kind of durability.*
- **DDS_Boolean direct_communication**
`<<extension>>` (p. 795) *Indicates whether or not a TRANSIENT or PERSISTENT **DDS_DataReader** (p. 615) should receive samples directly from a TRANSIENT or PERSISTENT **DDS_DataWriter** (p. 485)*
- **DDS_Long writer_depth**
`<<basic>>` (p. 783) `<<extension>>` (p. 795) *Indicates the number of samples per instance that a durable **DDS_↔DataWriter** (p. 485) will send to a late-joining **DDS_DataReader** (p. 615).*
- struct **DDS_PersistentStorageSettings storage_settings**
`<<extension>>` (p. 795) *Configures durable writer history and durable reader state.*

6.56.1 Detailed Description

<<*spec*>> (p. 790) This QoS policy specifies whether or not RTI Connex will store and deliver previously published data samples to new **DDS_DataReader** (p. 615) entities that join the network later.

Entity:

DDS_Topic (p. 175), **DDS_DataReader** (p. 615), **DDS_DataWriter** (p. 485)

Status:

DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 1229), **DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 1229)

Properties:

RxO (p. 1245) = YES

Changeable (p. 1245) = **UNTIL ENABLE** (p. 1245)

See also

DURABILITY_SERVICE (p. 1289)

6.56.2 Usage

It is possible for a **DDS_DataWriter** (p. 485) to start publishing data before all (or any) **DDS_DataReader** (p. 615) entities have joined the network.

Moreover, a **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615) 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 Connex will maintain internally. That is, RTI Connex 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. 1285) policy is set to **DDS_VOLATILE_DURABILITY_QOS** (p. 1287).

6.56.2.1 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. 1762) of **DDS_TRANSIENT_DURABILITY_QOS** (p. 1288) or **DDS_PERSISTENT_DURABILITY_QOS** (p. 1288) there is a corresponding "built-in" **DDS_DataReader** (p. 615) and **DDS_DataWriter** (p. 485) configured with the same DURABILITY kind. In other words, it is *as if* somewhere in the system, independent of the original **DDS_DataWriter** (p. 485), there is a built-in durable **DDS_DataReader** (p. 615) 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 **DDS_DataWriter** (p. 485) and **DDS_DataReader** (p. 615) entities. For each transient or persistent **DDS_Topic** (p. 175), the built-in fictitious Persistence Service **DDS_DataReader** (p. 615) and **DDS_DataWriter** (p. 485) have their QoS configured from the QoS of your application's **DDS_DataWriter** (p. 485) and **DDS_DataReader** (p. 615) entities that communicate on that **DDS_Topic** (p. 175).

For a given **DDS_Topic** (p. 175), the usual request/offered semantics apply to the matching between any **DDS_DataWriter** (p. 485) in the domain that writes the **DDS_Topic** (p. 175) and the built-in transient/persistent **DDS_DataReader** (p. 615) for that **DDS_Topic** (p. 175); similarly for the built-in transient/persistent **DDS_DataWriter** (p. 485) for a **DDS_Topic** (p. 175) and any **DDS_DataReader** (p. 615) for the **DDS_Topic** (p. 175). As a consequence, a **DDS_DataWriter** (p. 485) that has an incompatible QoS will not send its data to the *RTI Persistence Service*, and a **DDS_DataReader** (p. 615) that has an incompatible QoS will not get data from it.

Incompatibilities between local **DDS_DataReader** (p. 615) and **DDS_DataWriter** (p. 485) entities and the corresponding fictitious built-in transient/persistent entities cause the **DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 1229) and **DDS_OFFERED_INCOMPATIBLE_QOS_STATUS** (p. 1229) to change and the corresponding Listener invocations and/or signaling of **DDS_Condition** (p. 1370) objects as they would with your application's own entities.

The value of **DDS_DurabilityServiceQosPolicy::service_cleanup_delay** (p. 1765) 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 than **DDS_DurabilityServiceQosPolicy::service_cleanup_delay** (p. 1765) 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. 1208) are currently supported for the `service_cleanup_delay`)

The utility of **DDS_DurabilityServiceQosPolicy::service_cleanup_delay** (p. 1765) 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.

6.56.3 Compatibility

The value offered is considered compatible with the value requested if and only if the inequality *offered kind* $\geq$ *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. 1287) < **DDS_TRANSIENT_LOCAL_DURABILITY_QOS** (p. 1288) < **DDS_TRANSIENT_DURABILITY_QOS** (p. 1288) < **DDS_PERSISTENT_DURABILITY_QOS** (p. 1288).

6.56.4 Field Documentation

6.56.4.1 kind

DDS_DurabilityQosPolicyKind `DDS_DurabilityQosPolicy::kind`

<<**basic**>> (p. 783) <<**spec**>> (p. 790) The kind of durability.

[default] **DDS_VOLATILE_DURABILITY_QOS** (p. 1287)

Categories

Immutable (p. 914), **Matching** (p. 963), **Discovery Metadata** (p. 958), **Durability** (p. 860), **Data Availability** (p. 843)

6.56.4.2 direct_communication

DDS_Boolean `DDS_DurabilityQosPolicy::direct_communication`

<<**extension**>> (p. 795) Indicates whether or not a TRANSIENT or PERSISTENT **DDS_DataReader** (p. 615) should receive samples directly from a TRANSIENT or PERSISTENT **DDS_DataWriter** (p. 485)

When `direct_communication` is set to **DDS_BOOLEAN_TRUE** (p. 1201), a TRANSIENT or PERSISTENT **DDS_DataReader** (p. 615) will receive samples from both the original **DDS_DataWriter** (p. 485) configured with TRANSIENT or PERSISTENT durability and the **DDS_DataWriter** (p. 485) 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 **DDS_DataWriter** (p. 485) and the persistence service, the middleware will discard the duplicate.

When `direct_communication` is set to **DDS_BOOLEAN_FALSE** (p. 1201), a TRANSIENT or PERSISTENT **DDS_DataReader** (p. 615) will only receive samples from the **DDS_DataWriter** (p. 485) created by the persistence service. This brokered communication pattern provides a way to guarantee eventual consistency.

[default] **DDS_BOOLEAN_TRUE** (p. 1201)

Categories

Immutable (p. 914), **Discovery Metadata** (p. 958), **Durability** (p. 860), **Data Availability** (p. 843)

6.56.4.3 writer_depth

`DDS_Long DDS_DurabilityQosPolicy::writer_depth`

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Indicates the number of samples per instance that a durable **DDS_DataWriter** (p. 485) will send to a late-joining **DDS_DataReader** (p. 615).

The default value, **DDS_AUTO_WRITER_DEPTH** (p. 1288), makes this parameter equal to the following:

- **DDS_HistoryQosPolicy::depth** (p. 1807) if the history kind is **DDS_KEEP_LAST_HISTORY_QOS** (p. 1293).
- **DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1952) in the **DDS_ResourceLimitsQosPolicy** (p. 1949) if the history kind is **DDS_KEEP_ALL_HISTORY_QOS** (p. 1293).

The `writer_depth` must be $\leq$ **DDS_HistoryQosPolicy::depth** (p. 1807).

`writer_depth` applies only to non-volatile DataWriters (those for which the kind is **TRANSIENT_LOCAL**, **TRANSIENT**, or **PERSISTENT**).

When **DDS_BatchQosPolicy::enable** (p. 1552) 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. 1807).

Setting `writer_depth` on the DataReader side will be ignored.

[default] **DDS_AUTO_WRITER_DEPTH** (p. 1288)

Categories

Immutable (p. 914), **Durability** (p. 860), **Applied Per-Instance** (p. 956), **Data Availability** (p. 843)

6.56.4.4 storage_settings

`struct DDS_PersistentStorageSettings DDS_DurabilityQosPolicy::storage_settings`

<<**extension**>> (p. 795) 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. 1885) to **DDS_BOOLEAN_TRUE** (p. 1201).

Categories

Immutable (p. 914), **Durability** (p. 860), **Data Availability** (p. 843)

6.57 DDS_DurabilityServiceQosPolicy Struct Reference

<<*spec*>> (p. 790) Various settings to configure the external *RTI Persistence Service* used by RTI Connex for DataWriters with a **DDS_DurabilityQosPolicy** (p. 1759) setting of **DDS_PERSISTENT_DURABILITY_QOS** (p. 1288) or **DDS_TRANSIENT_DURABILITY_QOS** (p. 1288).

Data Fields

- struct **DDS_Duration_t service_cleanup_delay**
 <<*spec*>> (p. 790) Controls when the service is able to remove all information regarding a data instances.
- **DDS_HistoryQosPolicyKind history_kind**
 <<*spec*>> (p. 790) The kind of history to apply in recouping durable data.
- **DDS_Long history_depth**
 <<*spec*>> (p. 790) Setting to use for the **DDS_DurabilityQosPolicy::writer_depth** (p. 1762) when recouping durable data.
- **DDS_Long max_samples**
 <<*spec*>> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
- **DDS_Long max_instances**
 <<*spec*>> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.
- **DDS_Long max_samples_per_instance**
 <<*spec*>> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.

6.57.1 Detailed Description

<<*spec*>> (p. 790) Various settings to configure the external *RTI Persistence Service* used by RTI Connex for DataWriters with a **DDS_DurabilityQosPolicy** (p. 1759) setting of **DDS_PERSISTENT_DURABILITY_QOS** (p. 1288) or **DDS_TRANSIENT_DURABILITY_QOS** (p. 1288).

Entity:

DDS_Topic (p. 175), **DDS_DataWriter** (p. 485)

Properties:

RxO (p. 1245) = NO

Changeable (p. 1245) = **UNTIL ENABLE** (p. 1245)

See also

DURABILITY (p. 1285)

HISTORY (p. 1292)

RESOURCE_LIMITS (p. 1325)

6.57.2 Usage

When a DataWriter's **DDS_DurabilityQosPolicy::kind** (p. 1762) is **DDS_PERSISTENT_DURABILITY_QOS** (p. 1288) or **DDS_TRANSIENT_DURABILITY_QOS** (p. 1288), an external service, the *RTI Persistence Service*, is used to store and possibly forward the data sent by the **DDS_DataWriter** (p. 485) to **DDS_DataReader** (p. 615) 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 **DDS_DataWriter** (p. 485), such as how much data to store. For example, it configures the **DURABILITY** (p. 1285), **HISTORY** (p. 1292), and the **RESOURCE_LIMITS** (p. 1325) 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.

6.57.3 Field Documentation

6.57.3.1 service_cleanup_delay

```
struct DDS_Duration_t DDS_DurabilityServiceQosPolicy::service_cleanup_delay
```

<<*spec*>> (p. 790) 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. 1208) are currently supported.

[default] 0

Categories

Immutable (p. 914), **Discovery Metadata** (p. 958), **Durability** (p. 860), **Instance Lifecycle** (p. 867), **Data Availability** (p. 843)

6.57.3.2 history_kind

```
DDS_HistoryQosPolicyKind DDS_DurabilityServiceQosPolicy::history_kind
```

<<*spec*>> (p. 790) The kind of history to apply in recouping durable data.

[default] **DDS_KEEP_LAST_HISTORY_QOS** (p. 1293)

Categories

Immutable (p. 914), **Discovery Metadata** (p. 958), **Durability** (p. 860), **Applied Per-Instance** (p. 956), **Sample Memory Management** (p. 902)

6.57.3.3 history_depth

DDS_Long DDS_DurabilityServiceQosPolicy::history_depth

<<**spec**>> (p. 790) Setting to use for the **DDS_DurabilityQosPolicy::writer_depth** (p. 1762) when recouping durable data.

If the **DDS_HistoryQosPolicy::depth** (p. 1807) is set to a value lower than this value, **DDS_HistoryQosPolicy::depth** (p. 1807) will be set equal to the value of this field.

[default] **DDS_AUTO_WRITER_DEPTH** (p. 1288) (1)

Categories

Immutable (p. 914), **Discovery Metadata** (p. 958), **Durability** (p. 860), **Applied Per-Instance** (p. 956), **Sample Memory Management** (p. 902)

6.57.3.4 max_samples

DDS_Long DDS_DurabilityServiceQosPolicy::max_samples

<<**spec**>> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.

[default] **DDS_LENGTH_UNLIMITED** (p. 1326)

Categories

Immutable (p. 914), **Discovery Metadata** (p. 958), **Durability** (p. 860), **Sample Memory Management** (p. 902), **Middleware Memory Management** (p. 878), **Data Availability** (p. 843)

6.57.3.5 max_instances

DDS_Long DDS_DurabilityServiceQosPolicy::max_instances

<<**spec**>> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.

[default] **DDS_LENGTH_UNLIMITED** (p. 1326)

Categories

Immutable (p. 914), **Discovery Metadata** (p. 958), **Durability** (p. 860), **Instance Lifecycle** (p. 867), **Sample Memory Management** (p. 902), **Middleware Memory Management** (p. 878), **Data Availability** (p. 843)

6.57.3.6 max_samples_per_instance

DDS_Long DDS_DurabilityServiceQosPolicy::max_samples_per_instance

<<*spec*>> (p. 790) Part of resource limits QoS policy to apply when feeding a late joiner.

[default] **DDS_LENGTH_UNLIMITED** (p. 1326)

Categories

Immutable (p. 914), **Discovery Metadata** (p. 958), **Durability** (p. 860), **Instance Lifecycle** (p. 867), **Sample Memory Management** (p. 902), **Middleware Memory Management** (p. 878), **Data Availability** (p. 843)

6.58 DDS_Duration_t Struct Reference

Type for *duration* representation.

Data Fields

- **DDS_Long** *sec*
seconds
- **DDS_UnsignedLong** *nanosec*
nanoseconds

6.58.1 Detailed Description

Type for *duration* representation.

Represents a time interval.

Examples

HelloWorld_publisher.c, and **HelloWorld_subscriber.c**.

6.58.2 Field Documentation

6.58.2.1 sec

DDS_Long DDS_Duration_t::sec

seconds

Examples

HelloWorld_subscriber.c.

6.58.2.2 nanosec

DDS_UnsignedLong DDS_Duration_t::nanosec

nanoseconds

[range] [0,1000000000)

6.59 DDS_DynamicData Struct Reference

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

6.59.1 Detailed Description

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

Objects of type **DDS_DynamicData** (p. 1768) represent corresponding objects of the type identified by their **DDS_↔TypeCode** (p. 2071). 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. 1768) objects can represent any complex data type, including those of type kinds **DDS_TK_↔ARRAY** (p. 243), **DDS_TK_SEQUENCE** (p. 243), **DDS_TK_STRUCT** (p. 243), **DDS_TK_UNION** (p. 243), and **DDS_↔TK_VALUE** (p. 243). 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.

6.59.2 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;
};
```

6.59.2.1 Getting and Setting Union Discriminator Values

For unions (**DDS_TK_UNION** (p. 243)), 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. 335) 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. 243), about the member that the discriminator value selects, use the **DDS_DynamicData_get_member_info** (p. 347) 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. 347) will return **DDS_RETCODE_NO_DATA** (p. 1222).

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. 394) 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 member 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. 334) API.

Note: In past releases, accessing the discriminator value required passing in index 0 to the **DDS_DynamicData_get_member_info_by_index** (p. 348) operation and then looking at the populated `info.member_id`. That functionality has been deprecated and replaced with the **DDS_DynamicData_get_discriminator** (p. 335) operation.

6.59.2.2 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. 342) 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 theNestedType 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. 342) and **DDS_DynamicData_unbind_complex_member** (p. 344) 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. 399) 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]".

6.59.3 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. 1771)) and check the resulting length.
2. Get the length without getting the data itself: call **DDS_DynamicData_get_member_info** (p. 347) and check the resulting **DDS_DynamicDataMemberInfo::element_count** (p. 1779).

6.59.4 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.

6.59.4.1 Lifecycle and Utility Methods

Managing the lifecycle of **DDS_DynamicData** (p. 1768) objects is simple. You have two choices:

1. Usually, you will go through a **DDS_DynamicDataSupport** (p. 326) factory object, which will ensure that the type and property information for the new **DDS_DynamicData** (p. 1768) 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. 1768) 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 6.10 Lifecycle

DDS_DynamicDataSupport (p. 326)	DDS_DynamicData (p. 1768)
DDS_DynamicDataSupport_create_data (p. 435)	DDS_DynamicData_new (p. 327)
DDS_DynamicDataSupport_initialize_data (p. 436)	DDS_DynamicData_initialize (p. 327)
DDS_DynamicDataSupport_finalize_data (p. 436)	DDS_DynamicData_finalize (p. 328)
DDS_DynamicDataSupport_delete_data (p. 435)	DDS_DynamicData_delete (p. 329)

You can also copy **DDS_DynamicData** (p. 1768) objects:

- **DDS_DynamicData_copy** (p. 329)

You can test them for equality:

- **DDS_DynamicData_equal** (p. 329)

And you can print their contents:

- **DDS_DynamicData_print** (p. 336)
- **DDS_DynamicDataSupport_print_data** (p. 435)

6.59.4.2 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 Connex Modern C++ API reference.)

Table 6.11 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
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. 1768) 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. 1222). 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. 1768) 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. 1222).

Table 6.12 Basic Types

Get	Set
DDS_DynamicData_get_long (p. 351)	DDS_DynamicData_set_long (p. 394)
DDS_DynamicData_get_ulong (p. 352)	DDS_DynamicData_set_ulong (p. 395)
DDS_DynamicData_get_short (p. 351)	DDS_DynamicData_set_short (p. 394)
DDS_DynamicData_get_ushort (p. 353)	DDS_DynamicData_set_ushort (p. 396)
DDS_DynamicData_get_longlong (p. 358)	DDS_DynamicData_set_longlong (p. 400)
DDS_DynamicData_get_ulonglong (p. 359)	DDS_DynamicData_set_ulonglong (p. 401)
DDS_DynamicData_get_float (p. 354)	DDS_DynamicData_set_float (p. 396)
DDS_DynamicData_get_double (p. 355)	DDS_DynamicData_set_double (p. 397)

Get	Set
DDS_DynamicData_get_longdouble (p. 360)	DDS_DynamicData_set_longdouble (p. 402)
DDS_DynamicData_get_boolean (p. 356)	DDS_DynamicData_set_boolean (p. 398)
DDS_DynamicData_get_octet (p. 357)	DDS_DynamicData_set_octet (p. 399)
DDS_DynamicData_get_char (p. 356)	DDS_DynamicData_set_char (p. 399)
DDS_DynamicData_get_wchar (p. 361)	DDS_DynamicData_set_wchar (p. 402)
DDS_DynamicData_get_string (p. 363)	DDS_DynamicData_set_string (p. 405)
DDS_DynamicData_get_wstring (p. 364)	DDS_DynamicData_set_wstring (p. 405)

Table 6.13 Structures, Arrays, and Other Complex Types

Get	Set
DDS_DynamicData_get_complex_member (p. 365)	DDS_DynamicData_set_complex_member (p. 406)

Table 6.14 Arrays of Basic Types

Get	Set
DDS_DynamicData_get_long_array (p. 366)	DDS_DynamicData_set_long_array (p. 408)
DDS_DynamicData_get_ulong_array (p. 368)	DDS_DynamicData_set_ulong_array (p. 409)
DDS_DynamicData_get_short_array (p. 367)	DDS_DynamicData_set_short_array (p. 408)
DDS_DynamicData_get_ushort_array (p. 369)	DDS_DynamicData_set_ushort_array (p. 410)
DDS_DynamicData_get_longlong_array (p. 375)	DDS_DynamicData_set_longlong_array (p. 415)
DDS_DynamicData_get_ulonglong_array (p. 376)	DDS_DynamicData_set_ulonglong_array (p. 416)
DDS_DynamicData_get_float_array (p. 370)	DDS_DynamicData_set_float_array (p. 411)
DDS_DynamicData_get_double_array (p. 371)	DDS_DynamicData_set_double_array (p. 412)
DDS_DynamicData_get_longdouble_array (p. 377)	DDS_DynamicData_set_longdouble_array (p. 417)
DDS_DynamicData_get_boolean_array (p. 372)	DDS_DynamicData_set_boolean (p. 398)
DDS_DynamicData_get_octet_array (p. 374)	DDS_DynamicData_set_octet_array (p. 415)
DDS_DynamicData_get_char_array (p. 373)	DDS_DynamicData_set_char_array (p. 414)
DDS_DynamicData_get_wchar_array (p. 378)	DDS_DynamicData_set_wchar_array (p. 418)

Table 6.15 Sequences of Basic Types

Get	Set
DDS_DynamicData_get_long_seq (p. 381)	DDS_DynamicData_set_long_seq (p. 421)
DDS_DynamicData_get_ulong_seq (p. 383)	DDS_DynamicData_set_ulong_seq (p. 422)
DDS_DynamicData_get_short_seq (p. 382)	DDS_DynamicData_set_short_seq (p. 421)
DDS_DynamicData_get_ushort_seq (p. 383)	DDS_DynamicData_set_ushort_seq (p. 423)

Get	Set
DDS_DynamicData_get_longlong_seq (p. 389)	DDS_DynamicData_set_longlong_seq (p. 427)
DDS_DynamicData_get_ulonglong_seq (p. 389)	DDS_DynamicData_set_ulonglong_seq (p. 428)
DDS_DynamicData_get_float_seq (p. 384)	DDS_DynamicData_set_float_seq (p. 424)
DDS_DynamicData_get_double_seq (p. 385)	DDS_DynamicData_set_double_seq (p. 424)
DDS_DynamicData_get_longdouble_seq (p. 390)	DDS_DynamicData_set_longdouble_seq (p. 429)
DDS_DynamicData_get_boolean_seq (p. 386)	DDS_DynamicData_set_boolean_seq (p. 425)
DDS_DynamicData_get_octet_seq (p. 388)	DDS_DynamicData_set_octet_seq (p. 427)
DDS_DynamicData_get_char_seq (p. 387)	DDS_DynamicData_set_char_seq (p. 426)
DDS_DynamicData_get_wchar_seq (p. 391)	DDS_DynamicData_set_wchar_seq (p. 430)

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. 330)
- **DDS_DynamicData_clear_optional_member** (p. 330)

6.59.4.3 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. 344)
- **DDS_DynamicData_get_type_kind** (p. 345)
- **DDS_DynamicData_get_member_type** (p. 349)
- **DDS_DynamicData_get_member_info** (p. 347)
- **DDS_DynamicData_get_member_count** (p. 345)
- **DDS_DynamicData_get_member_info_by_index** (p. 348)
- **DDS_DynamicData_member_exists** (p. 345)
- **DDS_DynamicData_member_exists_in_type** (p. 346)
- **DDS_DynamicData_is_member_key** (p. 350)

6.59.4.4 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. 1768) 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. 1768) object as a view into an enclosing **DDS_DynamicData** (p. 1768) object.

- **DDS_DynamicData_bind_type** (p. 340)
- **DDS_DynamicData_unbind_type** (p. 341)
- **DDS_DynamicData_bind_complex_member** (p. 342)
- **DDS_DynamicData_unbind_complex_member** (p. 344)

6.59.4.5 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 `DynamicDataReader` 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_DynamicDataSerializationProperty_t::skip_deserialization` (p. 1783) to `DDS_BOOLEAN_TRUE` (p. 1201).

Once a sample is received, you can access the serialized buffer directly by using the `DDS_DynamicData_get_cdr_buffer` (p. 332). 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. 332) 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. 1768) object concurrently from multiple threads is *unsafe*.

Examples

`HelloWorldPlugin.c`.

6.60 DDS_DynamicDataInfo Struct Reference

A descriptor for a `DDS_DynamicData` (p. 1768) object.

Data Fields

- **DDS_Long member_count**

*The number of data members in this **DDS_DynamicData** (p. 1768) sample.*

- **DDS_Long stored_size**

*The number of bytes currently used to store the data of this **DDS_DynamicData** (p. 1768) sample.*

6.60.1 Detailed Description

A descriptor for a **DDS_DynamicData** (p. 1768) object.

See also

DDS_DynamicData_get_info (p. 340)

6.60.2 Field Documentation

6.60.2.1 member_count

DDS_Long `DDS_DynamicDataInfo::member_count`

The number of data members in this **DDS_DynamicData** (p. 1768) sample.

6.60.2.2 stored_size

DDS_Long `DDS_DynamicDataInfo::stored_size`

The number of bytes currently used to store the data of this **DDS_DynamicData** (p. 1768) sample.

6.61 DDS_DynamicDataJsonParserProperties_t Struct Reference

A collection of attributes used to configure **DDS_DynamicData** (p. 1768) objects.

6.61.1 Detailed Description

A collection of attributes used to configure **DDS_DynamicData** (p. 1768) objects.

To ensure that new objects are initialized to known values, assign them with the static initializer **DDS_DynamicDataProperty_t_INITIALIZER** (p. 324).

See also

DDS_DynamicDataProperty_t_INITIALIZER (p. 324)

6.62 DDS_DynamicDataMemberInfo Struct Reference

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

Data Fields

- **DDS_DynamicDataMemberId member_id**
*An integer that uniquely identifies the data member within this **DDS_DynamicData** (p. 1768) 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.

6.62.1 Detailed Description

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

See also

DDS_DynamicData_get_member_info (p. 347)

6.62.2 Field Documentation

6.62.2.1 member_id

DDS_DynamicDataMemberId DDS_DynamicDataMemberInfo::member_id

An integer that uniquely identifies the data member within this **DDS_DynamicData** (p. 1768) 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. 243)

6.62.2.2 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.

6.62.2.3 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. 346)

6.62.2.4 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. 2071) and getting the **DDS_TCKind** (p. 243) from there.

6.62.2.5 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.

6.62.2.6 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. 243) here.

6.63 DDS_DynamicDataProperty_t Struct Reference

A collection of attributes used to configure **DDS_DynamicData** (p. 1768) objects.

Data Fields

- **DDS_Long** **buffer_initial_size**
*The initial amount of memory used by the underlying **DDS_DynamicData** (p. 1768) buffer, in bytes.*
- **DDS_Long** **buffer_max_size**
*The maximum amount of memory that the underlying **DDS_DynamicData** (p. 1768) buffer may use, in bytes.*

6.63.1 Detailed Description

A collection of attributes used to configure **DDS_DynamicData** (p. 1768) objects.

To ensure that new objects are initialized to known values, assign them with the static initializer **DDS_DynamicDataProperty_t_INITIALIZER** (p. 324).

See also

DDS_DynamicDataProperty_t_INITIALIZER (p. 324)

6.63.2 Field Documentation

6.63.2.1 `buffer_initial_size`

DDS_Long `DDS_DynamicDataProperty_t::buffer_initial_size`

The initial amount of memory used by the underlying **DDS_DynamicData** (p. 1768) 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. 1780)

6.63.2.2 `buffer_max_size`

DDS_Long `DDS_DynamicDataProperty_t::buffer_max_size`

The maximum amount of memory that the underlying **DDS_DynamicData** (p. 1768) 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. 1780)

6.64 DDS_DynamicDataSeq Struct Reference

An ordered collection of **DDS_DynamicData** (p. 1768) elements.

6.64.1 Detailed Description

An ordered collection of **DDS_DynamicData** (p. 1768) elements.

Instantiates **FooSeq** (p. 2113) < **DDS_DynamicData** (p. 1768) > .

See also

FooSeq (p. 2113)

DDS_DynamicData (p. 1768)

6.65 DDS_DynamicDataTypeProperty_t Struct Reference

A collection of attributes used to configure **DDS_DynamicDataTypeSupport** (p. 326) objects.

Data Fields

- struct **DDS_DynamicDataProperty_t data**

*These properties will be provided to every new **DDS_DynamicData** (p. 1768) sample created from the **DDS_DynamicData↔**
DataTypeSupport (p. 326).*

- struct **DDS_DynamicDataTypeSerializationProperty_t serialization**

Properties that govern how the data of this type will be serialized on the network.

6.65.1 Detailed Description

A collection of attributes used to configure **DDS_DynamicDataTypeSupport** (p. 326) objects.

The properties of a **DDS_DynamicDataTypeSupport** (p. 326) object contain the properties that will be used to instantiate any samples created by that object.

6.65.2 Field Documentation

6.65.2.1 data

```
struct DDS_DynamicDataProperty_t DDS_DynamicDataTypeProperty_t::data
```

These properties will be provided to every new **DDS_DynamicData** (p. 1768) sample created from the **DDS_DynamicData↔TypeSupport** (p. 326).

These properties are also used to create the samples that are kept in the DataReader's queue.

6.65.2.2 serialization

```
struct DDS_DynamicDataTypeSerializationProperty_t DDS_DynamicDataTypeProperty_t::serialization
```

Properties that govern how the data of this type will be serialized on the network.

6.66 DDS_DynamicDataTypeSerializationProperty_t Struct Reference

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

Data Fields

- **DDS_Boolean use_42e_compatible_alignment**
[No longer supported] Use RTI Connex 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. 1780) 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. 1780) 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.

6.66.1 Detailed Description

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

6.66.2 Field Documentation

6.66.2.1 use_42e_compatible_alignment

DDS_Boolean DDS_DynamicDataTypeSerializationProperty_t::use_42e_compatible_alignment

[No longer supported] Use RTI Connex 4.2e-compatible alignment for large primitive types.

In RTI Connex 4.2e, the default alignment for large primitive types – **DDS_LongLong** (p. 1203), **DDS_UnsignedLong** (p. 1203), **DDS_Double** (p. 1203), and **DDS_LongDouble** (p. 1204) – was not RTPS-compliant. This compatibility mode allows applications targeting post-4.2e versions of RTI Connex 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 Connex 4.2e.

6.66.2.2 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. 1780) 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. 2071), whichever is smaller.

6.66.2.3 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. 1780) 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.

6.66.2.4 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. 1780).

6.66.2.5 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. 1201) 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 function **DDS_DynamicData_get_cdr_buffer** (p. 332). 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. 1201), 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. 333)

DDS_DynamicData_get_cdr_buffer (p. 332)

DDS_DynamicData_set_cdr_buffer (p. 332)

[default] **DDS_BOOLEAN_FALSE** (p. 1201)

6.67 DDS_EndpointGroup_t Struct Reference

Specifies a group of endpoints that can be collectively identified by a name and satisfied by a quorum.

Data Fields

- `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.

6.67.1 Detailed Description

Specifies a group of endpoints that can be collectively identified by a name and satisfied by a quorum.

6.67.2 Field Documentation

6.67.2.1 role_name

```
char* DDS_EndpointGroup_t::role_name
```

Defines the role name of the endpoint group.

If used in the **DDS_AvailabilityQosPolicy** (p. 1547) on a **DDS_DataWriter** (p. 485), it specifies the name that identifies a Durable Subscription.

The role name can be at most 255 characters in length.

6.67.2.2 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. 1547) on a **DDS_DataWriter** (p. 485), it specifies the number of Data↔Readers that must acknowledge a sample before the sample is considered to be acknowledged by the Durable Subscription.

6.68 DDS_EndpointGroupSeq Struct Reference

A sequence of **DDS_EndpointGroup_t** (p. 1784).

6.68.1 Detailed Description

A sequence of **DDS_EndpointGroup_t** (p. 1784).

In the context of Collaborative DataWriters, it can be used by a **DDS_DataReader** (p. 615) to define a group of remote DataWriters that the **DDS_DataReader** (p. 615) 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. 1013) **FooSeq** (p. 2113)

See also

DDS_EndpointGroup_t (p. 1784)

6.69 DDS_EndpointTrustAlgorithmInfo Struct Reference

Trust Plugins algorithm information associated with the discovered endpoint.

Data Fields

- **DDS_EndpointTrustInterceptorAlgorithmInfo interceptor**

Information regarding algorithms for interception of data and metadata exchanged by the endpoint.

6.69.1 Detailed Description

Trust Plugins algorithm information associated with the discovered endpoint.

6.69.2 Field Documentation

6.69.2.1 interceptor

DDS_EndpointTrustInterceptorAlgorithmInfo DDS_EndpointTrustAlgorithmInfo::interceptor

Information regarding algorithms for interception of data and metadata exchanged by the endpoint.

6.70 DDS_EndpointTrustInterceptorAlgorithmInfo Struct Reference

Trust Plugins interception algorithm information associated with the discovered endpoint.

Data Fields

- **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.

6.70.1 Detailed Description

Trust Plugins interception algorithm information associated with the discovered endpoint.

6.70.2 Field Documentation

6.70.2.1 required_mask

`DDS_TrustAlgorithmSet DDS_EndpointTrustInterceptorAlgorithmInfo::required_mask`

Trust Plugins algorithms used for interception of data and metadata exchanged by the endpoint.

6.70.2.2 supported_mask

`DDS_TrustAlgorithmSet DDS_EndpointTrustInterceptorAlgorithmInfo::supported_mask`

Trust Plugins algorithms supported for interception of data and metadata exchanged by the endpoint.

6.71 DDS_EndpointTrustProtectionInfo Struct Reference

Trust Plugins Protection information associated with the discovered endpoint.

Data Fields

- **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.

6.71.1 Detailed Description

Trust Plugins Protection information associated with the discovered endpoint.

6.71.2 Field Documentation

6.71.2.1 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](#).

6.71.2.2 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](#).

6.72 DDS_EntityFactoryQosPolicy Struct Reference

<<*spec*>> (p. 790) A QoS policy for all **DDS_Entity** (p. 1360) types that can act as factories for one or more other **DDS_Entity** (p. 1360) types.

Data Fields

- **DDS_Boolean autoenable_created_entities**

<<*basic*>> (p. 783) <<*spec*>> (p. 790) *Specifies whether the entity acting as a factory automatically enables the instances it creates.*

6.72.1 Detailed Description

<<*spec*>> (p. 790) A QoS policy for all **DDS_Entity** (p. 1360) types that can act as factories for one or more other **DDS_Entity** (p. 1360) types.

Entity:

DDS_DomainParticipantFactory (p. 32), **DDS_DomainParticipant** (p. 77), **DDS_Publisher** (p. 443), **DDS_Subscriber** (p. 572)

Properties:

RxO (p. 1245) = NO

Changeable (p. 1245) = **YES** (p. 1245)

6.72.2 Usage

This policy controls the behavior of the **DDS_Entity** (p. 1360) 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 **DDS_DataReader** (p. 615) is created in an enabled state, its existence is immediately propagated for discovery and the **DDS_DataReader** (p. 615) object's listener called as soon as data is received. The initialization process for an application may extend beyond the creation of the **DDS_DataReader** (p. 615), and thus, it may not be desirable for the **DDS_DataReader** (p. 615) 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.

6.72.3 Field Documentation

6.72.3.1 autoenable_created_entities

DDS_Boolean DDS_EntityFactoryQosPolicy::autoenable_created_entities

<<**basic**>> (p. 783) <<**spec**>> (p. 790) 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. 1201) indicates that the factory `create_<entity>` operation(s) will automatically invoke the **DDS_Entity_enable** (p. 1363) operation each time a new **DDS_Entity** (p. 1360) is created. Therefore, the **DDS_Entity** (p. 1360) returned by `create_<entity>` will already be enabled. A setting of **DDS_BOOLEAN_FALSE** (p. 1201) indicates that the **DDS_Entity** (p. 1360) will not be automatically enabled. Your application will need to call **DDS_Entity_enable** (p. 1363) itself.

The default setting of `autoenable_created_entities = DDS_BOOLEAN_TRUE` (p. 1201) means that, by default, it is not necessary to explicitly call **DDS_Entity_enable** (p. 1363) on newly created entities.

[default] **DDS_BOOLEAN_TRUE** (p. 1201)

Categories

Mutable (p. 949), **Application Startup** (p. 838)

6.73 DDS_EntityNameQosPolicy Struct Reference

<<*extension*>> (p. 795) Assigns a name and a role name to a **DDS_DomainParticipant** (p. 77), **DDS_Publisher** (p. 443), **DDS_Subscriber** (p. 572), **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615). Except for **DDS_Publisher** (p. 443) and **DDS_Subscriber** (p. 572), these names will be visible during the discovery process and in RTI tools to help you visualize and debug your system.

Data Fields

- char * **name**
<<*basic*>> (p. 783) <<*extension*>> (p. 795) *The name of the entity.*
- char * **role_name**
<<*extension*>> (p. 795) *The entity role name.*

6.73.1 Detailed Description

<<*extension*>> (p. 795) Assigns a name and a role name to a **DDS_DomainParticipant** (p. 77), **DDS_Publisher** (p. 443), **DDS_Subscriber** (p. 572), **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615). Except for **DDS_Publisher** (p. 443) and **DDS_Subscriber** (p. 572), these names will be visible during the discovery process and in RTI tools to help you visualize and debug your system.

Entity:

DDS_DomainParticipant (p. 77), **DDS_Subscriber** (p. 572), **DDS_Publisher** (p. 443), **DDS_DataReader** (p. 615), **DDS_DataWriter** (p. 485)

Properties:

RxO (p. 1245) = NO;
Changeable (p. 1245) = **UNTIL ENABLE** (p. 1245)

6.73.2 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. 1528) or **DDS_String_dup** (p. 1528).

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.

6.73.3 Field Documentation

6.73.3.1 name

```
char* DDS_EntityNameQosPolicy::name
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) The name of the entity.

[default] null

[range] Null terminated string with length not exceeding 255. It can be null.

Categories

Immutable (p. 914), **Debuggability** (p. 847), **Discovery Metadata** (p. 958)

6.73.3.2 role_name

```
char* DDS_EntityNameQosPolicy::role_name
```

<<**extension**>> (p. 795) The entity role name.

With Durable Subscriptions this name is used to specify to which Durable Subscription the **DDS_DataReader** (p. 615) belongs.

With Collaborative DataWriters this name is used to specify to which endpoint group the **DDS_DataWriter** (p. 485) belongs.

[range] Null terminated string with length not exceeding 255. It can be null.

[default] null

Categories

Immutable (p. 914), **Debuggability** (p. 847), **Discovery Metadata** (p. 958), **Durability** (p. 860)

6.74 DDS_EnumMember Struct Reference

A description of a member of an enumeration.

Data Fields

- char * **name**
The name of the enumeration member.
- **DDS_Long ordinal**
The value associated the the enumeration member.

6.74.1 Detailed Description

A description of a member of an enumeration.

See also

DDS_EnumMemberSeq (p. 1792)

DDS_TypeCodeFactory_create_enum_tc (p. 296)

6.74.2 Field Documentation

6.74.2.1 name

```
char* DDS_EnumMember::name
```

The name of the enumeration member.

Cannot be NULL.

6.74.2.2 ordinal

```
DDS_Long DDS_EnumMember::ordinal
```

The value associated the the enumeration member.

6.75 DDS_EnumMemberSeq Struct Reference

Defines a sequence of enumerator members.

6.75.1 Detailed Description

Defines a sequence of enumerator members.

See also

DDS_EnumMember (p. 1791)

FooSeq (p. 2113)

DDS_TypeCodeFactory_create_enum_tc (p. 296)

6.76 DDS_EventQosPolicy Struct Reference

<<*extension*>> (p. 795) Settings for event.

Data Fields

- struct **DDS_ThreadSettings_t** **thread**
 <<*extension*>> (p. 795) *Event thread QoS.*
- **DDS_Long** **initial_count**
 <<*extension*>> (p. 795) *The initial number of events.*
- **DDS_Long** **max_count**
 <<*extension*>> (p. 795) *The maximum number of events.*

6.76.1 Detailed Description

<<*extension*>> (p. 795) Settings for event.

In a **DDS_DomainParticipant** (p. 77), 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 **DDS_DomainParticipant** (p. 77) 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 **DDS_DomainParticipant** (p. 77) from dynamically allocating memory for the event thread after initialization.

This QoS policy is an extension to the DDS standard.

Entity:

DDS_DomainParticipant (p. 77)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

6.76.2 Field Documentation

6.76.2.1 thread

```
struct DDS_ThreadSettings_t DDS_EventQosPolicy::thread
```

<<**extension**>> (p. 795) 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. 1237) | **DDS_THREAD_SETTINGS_STUDIO** (p. 1237)

Categories

Immutable (p. 914), **Concurrency** (p. 839)

6.76.2.2 initial_count

```
DDS_Long DDS_EventQosPolicy::initial_count
```

<<**extension**>> (p. 795) The initial number of events.

[default] 256

[range] [1, 1 million], <= max_count

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.76.2.3 max_count

DDS_Long DDS_EventQosPolicy::max_count

<<**extension**>> (p. 795) 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. 1326)

[range] [1, 1 million] or **DDS_LENGTH_UNLIMITED** (p. 1326), >= initial_count

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.77 DDS_ExpressionProperty Struct Reference

Provides additional information about the filter expression passed to **DDS_ContentFilter::writer_compile** (p. 1573).

Data Fields

- **DDS_Boolean key_only_filter**
Indicates if the filter expression is based only on key fields. In this case, RTI Connexx 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.

6.77.1 Detailed Description

Provides additional information about the filter expression passed to **DDS_ContentFilter::writer_compile** (p. 1573).

It is used by the filter implementation to indicate to the middleware whether or not the **DDS_ContentFilter** (p. 1571) will cache the result of filter evaluation.

6.77.2 Field Documentation

6.77.2.1 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 Connexx itself can cache the filtering results.

When this field is set to **DDS_BOOLEAN_TRUE** (p. 1201), it indicates to RTI Connexx that the filter expression is based only on key fields.

6.77.2.2 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. 1201), RTI Connex will do no caching or explicit filter evaluation for the associated **DDS_DataReader** (p. 615). Instead, it will rely on the filter implementation to provide appropriate results.

6.78 DDS_FilterSampleInfo Struct Reference

Provides meta information associated with the sample.

Data Fields

- 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.

6.78.1 Detailed Description

Provides meta information associated with the sample.

This can be used by a content filter to perform filtering on meta data.

6.78.2 Field Documentation

6.78.2.1 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 **DDS_↵_DataWriter** (p. 485) 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. 2102) when writing a sample using the function **FooDataWriter_write_w_params** (p. 501).

Default value **DDS_UNKNOWN_SAMPLE_IDENTITY** (p. 1392) denotes a sample not associated with any other sample.

A **DDS_DataReader** (p. 615) can inspect the related sample identity of a received sample by accessing the fields **DDS_SampleInfo::related_original_publication_virtual_guid** (p. 1993) and **DDS_SampleInfo::related_original_↵_publication_virtual_sequence_number** (p. 1994).

6.78.2.2 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. 2105) when writing a sample using the function **FooDataWriter_write_w_params** (p. 501).

Default value **DDS_GUID_UNKNOWN** (p. 1213) denotes a sample as lacking a related data source.

A **DDS_DataReader** (p. 615) can inspect the `related_source_guid` of a received sample by accessing the field **DDS_↵_SampleInfo::related_source_guid** (p. 1994).

6.78.2.3 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. 2105) when writing a sample using the function **FooDataWriter_write_w_params** (p. 501).

Default value **DDS_GUID_UNKNOWN** (p. 1213) denotes a sample without an associated DataReader.

A **DDS_DataReader** (p. 615) can inspect the `related_reader_guid` of a received sample by accessing the field **DDS_↵_SampleInfo::related_source_guid** (p. 1994).

6.78.2.4 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. 2102) when writing a sample using the function **FooDataWriter_write_w_params** (p. 501).

Default value **DDS_UNKNOWN_SAMPLE_IDENTITY** (p. 1392) 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 **DDS_DataReader** (p. 615) can inspect the identity of a received sample by accessing the fields **DDS_SampleInfo::original_publication_virtual_guid** (p. 1993) and **DDS_SampleInfo::original_publication_virtual_sequence_number** (p. 1993).

6.78.2.5 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. 2104) when writing a sample using the function **FooDataWriter_write_w_params** (p. 501).

Default value **DDS_GUID_UNKNOWN** (p. 1213) denotes a sample without a logical data source associated with it.

A **DDS_DataReader** (p. 615) can inspect the logical data source of a sample by accessing the field **DDS_SampleInfo::source_guid** (p. 1994).

6.79 DDS_FloatSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_Float** (p. 1203) >

6.79.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_Float** (p. 1203) >

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_Float (p. 1203)

FooSeq (p. 2113)

6.80 DDS_FlowControllerProperty_t Struct Reference

Determines the flow control characteristics of the **DDS_FlowController** (p. 559).

Data Fields

- **DDS_FlowControllerSchedulingPolicy** `scheduling_policy`
Scheduling policy.
- struct **DDS_FlowControllerTokenBucketProperty_t** `token_bucket`
Settings for the token bucket.

6.80.1 Detailed Description

Determines the flow control characteristics of the **DDS_FlowController** (p. 559).

The flow control characteristics shape the network traffic by determining how often and in what order associated asynchronous **DDS_DataWriter** (p. 485) instances are serviced and how much data they are allowed to send.

Note that these settings apply directly to the **DDS_FlowController** (p. 559), and do not depend on the number of **DDS_DataWriter** (p. 485) instances the **DDS_FlowController** (p. 559) is servicing. For instance, the specified flow rate does *not* double simply because two **DDS_DataWriter** (p. 485) instances are waiting to write.

Entity:

DDS_FlowController (p. 559)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245) for **DDS_FlowControllerProperty_t::scheduling_policy** (p. 1799), **YES** (p. 1245) for **DDS_FlowControllerProperty_t::token_bucket** (p. 1800). However, the special value of **DDS_DURATION_INFINITE** (p. 1208) as **DDS_FlowControllerTokenBucketProperty_t::period** (p. 1802) is strictly used to create an *on-demand* **DDS_FlowController** (p. 559). The token period cannot toggle from an infinite to finite value (or vice versa). It can, however, change from one finite value to another.

6.80.2 Field Documentation

6.80.2.1 scheduling_policy

DDS_FlowControllerSchedulingPolicy DDS_FlowControllerProperty_t::scheduling_policy

Scheduling policy.

Determines the scheduling policy for servicing the **DDS_DataWriter** (p. 485) instances associated with the **DDS_FlowController** (p. 559).

[default] **DDS_EDF_FLOW_CONTROLLER_SCHED_POLICY** (p. 560)

6.80.2.2 token_bucket

struct DDS_FlowControllerTokenBucketProperty_t DDS_FlowControllerProperty_t::token_bucket

Settings for the token bucket.

6.81 DDS_FlowControllerTokenBucketProperty_t Struct Reference

DDS_FlowController (p. 559) uses the popular token bucket approach for open loop network flow control. The flow control characteristics are determined by the token bucket properties.

Data Fields

- **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.

6.81.1 Detailed Description

DDS_FlowController (p. 559) 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:

DDS_FlowController (p. 559)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **YES** (p. 1245). However, the special value of **DDS_DURATION_INFINITE** (p. 1208) as **DDS_FlowControllerTokenBucketProperty_t::period** (p. 1802) is strictly used to create an *on-demand* **DDS_FlowController** (p. 559). The token period cannot toggle from an infinite to finite value (or vice versa). It can, however, change from one finite value to another.

6.81.2 Field Documentation

6.81.2.1 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. 1802), determines the maximum allowable data burst.

Use **DDS_LENGTH_UNLIMITED** (p. 1326) to allow accumulation of an unlimited amount of tokens (and therefore potentially an unlimited burst size).

[default] **DDS_LENGTH_UNLIMITED** (p. 1326)

[range] [1, **DDS_LENGTH_UNLIMITED** (p. 1326)]

6.81.2.2 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.

DDS_FlowController (p. 559) 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 **DDS_DataWriter** (p. 485) instances based on the **DDS_FlowController**↔**Property_t::scheduling_policy** (p. 1799).

Use **DDS_LENGTH_UNLIMITED** (p. 1326) to add the maximum number of tokens allowed by **DDS_FlowController**↔**TokenBucketProperty_t::max_tokens** (p. 1801).

[default] **DDS_LENGTH_UNLIMITED** (p. 1326)

[range] [1, **DDS_LENGTH_UNLIMITED** (p. 1326)]

6.81.2.3 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.

DDS_FlowController (p. 559) 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. 1326) 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. 1326)]

6.81.2.4 period

```
struct DDS_Duration_t DDS_FlowControllerTokenBucketProperty_t::period
```

Period for adding tokens to and removing tokens from the bucket.

DDS_FlowController (p. 559) 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. 1208) can be used to create an *on-demand* **DDS_FlowController** (p. 559), for which tokens are no longer replenished periodically. Instead, tokens must be added explicitly by calling **DDS_FlowController_trigger_flow** (p. 563). This external trigger adds **DDS_FlowControllerTokenBucketProperty_t::tokens_added_per_period** (p. 1801) 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. 1208)

6.81.2.5 bytes_per_token

DDS_Long DDS_FlowControllerTokenBucketProperty_t::bytes_per_token

Maximum number of bytes allowed to send for each token available.

DDS_FlowController (p. 559) 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 **DDS_DataWriter** (p. 485). That is, in cases where **DDS_FlowControllerTokenBucketProperty_t::bytes_per_token** (p. 1802) 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. 1799)).

Where fragmentation is required, the fragment size will be **DDS_FlowControllerTokenBucketProperty_t::bytes_per_token** (p. 1802) or the minimum largest message size across all transports installed with the **DDS_DataWriter** (p. 485), whichever is less.

Use **DDS_LENGTH_UNLIMITED** (p. 1326) to indicate that an unlimited number of bytes can be transmitted per token. In other words, a single token allows the recipient **DDS_DataWriter** (p. 485) 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. 1326)

[range] [1024, **DDS_LENGTH_UNLIMITED** (p. 1326)]

6.82 DDS_GroupDataQosPolicy Struct Reference

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 165) during discovery.

Data Fields

- struct **DDS_OctetSeq** value
 <<*spec*>> (p. 790) a sequence of octets

6.82.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 165) during discovery.

Entity:

DDS_Publisher (p. 443), **DDS_Subscriber** (p. 572)

Properties:

RxO (p. 1245) = NO

Changeable (p. 1245) = YES (p. 1245)

See also

DDS_DomainParticipant_get_builtin_subscriber (p. 129)

6.82.2 Usage

The additional information is attached to a **DDS_Publisher** (p. 443) or **DDS_Subscriber** (p. 572). This extra data is not used by RTI Connexx itself. When a remote application discovers the **DDS_Publisher** (p. 443) or **DDS_Subscriber** (p. 572), 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. 2040) and **DDS_UserDataQosPolicy** (p. 2086), 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 **DDS_DataReaderListener** (p. 1593), **DDS_DataWriterListener** (p. 1644) and operations such as **DDS_DomainParticipant_ignore_publication** (p. 132) and **DDS_DomainParticipant_ignore_subscription** (p. 133), 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. 1881), 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 Connexx stores the data placed in this policy in pre-allocated pools. It is therefore necessary to configure RTI Connexx 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. 1746) and **DDS_DomainParticipantResourceLimitsQosPolicy::subscriber_group_data_max_length** (p. 1747).

6.82.3 Field Documentation

6.82.3.1 value

```
struct DDS_OctetSeq DDS_GroupDataQosPolicy::value
```

<<*spec*>> (p. 790) a sequence of octets

[default] Empty (zero-sized)

[range] Octet sequence of length [0,max_length]

Categories

Mutable (p. 949), **Application-Defined Metadata** (p. 836), **Discovery Metadata** (p. 958)

6.83 DDS_GUID_t Struct Reference

Type for *GUID* (Global Unique Identifier) representation.

Data Fields

- **DDS_Octet value** [DDS_GUID_LENGTH]
A 16 byte array containing the GUID value.

6.83.1 Detailed Description

Type for *GUID* (Global Unique Identifier) representation.

Represents a 128 bit GUID.

Alternative representation of **DDS RTPS_GUID_t** (p. 1954). Memory and wire representation for this type is the same as the one for **DDS RTPS_GUID_t** (p. 1954).

6.83.2 Field Documentation

6.83.2.1 value

DDS_Octet DDS_GUID_t::value [DDS_GUID_LENGTH]

A 16 byte array containing the GUID value.

6.84 DDS_HistoryQosPolicy Struct Reference

<<*spec*>> (p. 790) Specifies the behavior of RTI Connex 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.

Data Fields

- **DDS_HistoryQosPolicyKind kind**
<<*basic*>> (p. 783) <<*spec*>> (p. 790) Specifies the kind of history to be kept.
- **DDS_Long depth**
<<*basic*>> (p. 783) <<*spec*>> (p. 790) Specifies the number of samples per instance to be kept, when the *kind* is **DDS_KEEP_LAST_HISTORY_QOS** (p. 1293)

6.84.1 Detailed Description

<<*spec*>> (p. 790) Specifies the behavior of RTI Connex 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 be stored by RTI Connex for a **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615). It controls whether RTI Connex 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 **DDS_DataWriter** (p. 485) on behalf of existing **DDS_DataReader** (p. 615) entities. The behavior with regards to a **DDS_DataReader** (p. 615) entities discovered after a sample is written is controlled by the **DURABILITY** (p. 1285) policy.

On the subscribing side, this QoS policy controls the samples that should be maintained until the application "takes" them from RTI Connex.

Entity:

DDS_Topic (p. 175), **DDS_DataReader** (p. 615), **DDS_DataWriter** (p. 485)

Properties:

RxO (p. 1245) = NO

Changeable (p. 1245) = **UNTIL ENABLE** (p. 1245)

See also

DDS_ReliabilityQosPolicy (p. 1936)

DDS_HistoryQosPolicy (p. 1805)

6.84.2 Usage

This policy controls the behavior of RTI Connex when the value of an instance changes before it is finally communicated to **DDS_DataReader** (p. 615) entities.

When a **DDS_DataWriter** (p. 485) sends data, or a **DDS_DataReader** (p. 615) 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. 1936) by controlling whether RTI Connex guarantees that *all* of the sent data is received (**DDS_KEEP_ALL_HISTORY_QOS** (p. 1293)) or if only the last N data values sent are guaranteed to be received (**DDS_KEEP_LAST_HISTORY_QOS** (p. 1293))—this is a reduced level of reliability.

The amount of data that is sent to new DataReaders who have configured their **DDS_DurabilityQosPolicy** (p. 1759) to receive previously published data is controlled by **DDS_DurabilityQosPolicy::writer_depth** (p. 1762), 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. 1949).

If `kind` is **DDS_KEEP_LAST_HISTORY_QOS** (p. 1293) (the default), then RTI Connexx 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 Connexx 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. 1293), then RTI Connexx will attempt to maintain and deliver all the values of the instance to existing subscribers. The resources that RTI Connexx can use to keep this history are limited by the settings of the **RESOURCE_LIMITS** (p. 1325). If the limit is reached, then the behavior of RTI Connexx will depend on the **RELIABILITY** (p. 1321). If the Reliability `kind` is **DDS_BEST_EFFORT_RELIABILITY_QOS** (p. 1323), then the old values will be discarded. If Reliability `kind` is **RELIABLE**, then RTI Connexx will block the **DDS_DataWriter** (p. 485) until it can deliver the necessary old values to all subscribers.

6.84.3 Consistency

This QoS policy's `depth` must be consistent with the **RESOURCE_LIMITS** (p. 1325) `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. 1949)

6.84.4 Field Documentation

6.84.4.1 kind

DDS_HistoryQosPolicyKind `DDS_HistoryQosPolicy::kind`

<<**basic**>> (p. 783) <<**spec**>> (p. 790) 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. 1815) expiration, replacement because the **DDS_HistoryQosPolicy::depth** (p. 1807) has been exceeded, or one of the **DDS_ResourceLimitsQosPolicy** (p. 1949) settings has been exceeded.

[default] **DDS_KEEP_LAST_HISTORY_QOS** (p. 1293)

Categories

Immutable (p. 914), **Applied Per-Instance** (p. 956), **Reliability** (p. 895), **Data Availability** (p. 843)

6.84.4.2 depth

DDS_Long DDS_HistoryQosPolicy::depth

<<*basic*>> (p. 783) <<*spec*>> (p. 790) Specifies the number of samples per instance to be kept, when the `kind` is **DDS_KEEP_LAST_HISTORY_QOS** (p. 1293)

If a value other than 1 (the default) is specified, it should be consistent with the settings of the **RESOURCE_LIMITS** (p. 1325) policy. That is:

depth <= **DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1952)

When the `kind` is **DDS_KEEP_ALL_HISTORY_QOS** (p. 1293), the depth has no effect. Its implied value is `infinity` (in practice limited by the settings of the **RESOURCE_LIMITS** (p. 1325) 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. 1762), not this depth.

[default] 1

[range] [1,100 million], <= **DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1952)

Categories

Immutable (p. 914), **Applied Per-Instance** (p. 956), **Reliability** (p. 895), **Data Availability** (p. 843)

6.85 DDS_InconsistentTopicStatus Struct Reference

DDS_INCONSISTENT_TOPIC_STATUS (p. 1228)

Data Fields

- **DDS_Long total_count**

*Total cumulative count of the pairs (**DDS_DataReader** (p. 615), **DDS_DataWriter** (p. 485)) whose topic names match the **DDS_Topic** (p. 175) to which this status is attached and whose types are inconsistent according to the rules defined in **DDS_TypeConsistencyEnforcementQosPolicy** (p. 2076).*

- **DDS_Long total_count_change**

*The incremental number of inconsistent pairs (**DDS_DataReader** (p. 615), **DDS_DataWriter** (p. 485)) for the **DDS_Topic** (p. 175) to which this status is attached, that have been discovered since the last time this status was read.*

6.85.1 Detailed Description

DDS_INCONSISTENT_TOPIC_STATUS (p. 1228)

Entity:

DDS_Topic (p. 175)

Listener:

DDS_TopicListener (p. 2041)

Every time a **DDS_DataReader** (p. 615) and **DDS_DataWriter** (p. 485) with the same **DDS_Topic** (p. 175) do not match because the type-consistency enforcement policy fails, the inconsistent topic status is updated for the **DDS_Topic** (p. 175).

See also

DDS_TypeConsistencyEnforcementQosPolicy (p. 2076)

6.85.2 Field Documentation

6.85.2.1 total_count

DDS_Long DDS_InconsistentTopicStatus::total_count

Total cumulative count of the pairs (**DDS_DataReader** (p. 615), **DDS_DataWriter** (p. 485)) whose topic names match the **DDS_Topic** (p. 175) to which this status is attached and whose types are inconsistent according to the rules defined in **DDS_TypeConsistencyEnforcementQosPolicy** (p. 2076).

6.85.2.2 total_count_change

DDS_Long DDS_InconsistentTopicStatus::total_count_change

The incremental number of inconsistent pairs (**DDS_DataReader** (p. 615), **DDS_DataWriter** (p. 485)) for the **DDS_Topic** (p. 175) to which this status is attached, that have been discovered since the last time this status was read.

6.86 DDS_InstanceHandleSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_InstanceHandle_t** (p. 214) > .

6.86.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_InstanceHandle_t** (p. 214) > .

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_InstanceHandle_t (p. 214)

FooSeq (p. 2113)

6.87 DDS_Int8Seq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_Int8** (p. 1202) >

6.87.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_Int8** (p. 1202) >

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_Int8 (p. 1202)

FooSeq (p. 2113)

6.88 DDS_InvalidLocalAdvanceNoticeStatus Struct Reference

DDS_INVALID_LOCAL_ADVANCE_NOTICE_STATUS (p. 1232)

Data Fields

- 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. 1811) code.*

6.88.1 Detailed Description

DDS_INVALID_LOCAL_ADVANCE_NOTICE_STATUS (p. 1232)

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.

6.88.2 Field Documentation

6.88.2.1 expiration_time

```
struct DDS_Time_t DDS_InvalidLocalAdvanceNoticeStatus::expiration_time
```

The time when the local artifact will expire.

6.88.2.2 expiration_code

```
DDS_UnsignedLong DDS_InvalidLocalAdvanceNoticeStatus::expiration_code
```

The reason (a numeric code) for the local artifact to expire.

6.88.2.3 expiration_message

```
const char* DDS_InvalidLocalAdvanceNoticeStatus::expiration_message
```

A descriptive message for the `DDS_InvalidLocalAdvanceNoticeStatus::expiration_code` (p. 1811) code.

6.89 DDS_KeyedOctets Struct Reference

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

Data Fields

- char * **key**
Instance key associated with the specified value.
- int **length**
Number of octets to serialize.
- unsigned char * **value**
DDS_Octets (p. 1859) array value.

6.89.1 Detailed Description

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

6.90 DDS_KeyedOctetsSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_KeyedOctets** (p. 1811) >.

6.90.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_KeyedOctets** (p. 1811) >.

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_KeyedOctets (p. 1811)

6.91 DDS_KeyedOctetsTypeSupport Struct Reference

<<*interface*>> (p. 1013) **DDS_KeyedOctets** (p. 1811) type support.

6.91.1 Detailed Description

<<*interface*>> (p. 1013) **DDS_KeyedOctets** (p. 1811) type support.

6.92 DDS_KeyedString Struct Reference

Keyed string built-in type.

Data Fields

- char * **key**
Instance key associated with the specified value.
- char * **value**
String value.

6.92.1 Detailed Description

Keyed string built-in type.

6.93 DDS_KeyedStringSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_KeyedString** (p. 1812) > .

6.93.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_KeyedString** (p. 1812) > .

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_KeyedString (p. 1812)

6.94 DDS_KeyedStringTypeSupport Struct Reference

<<*interface*>> (p. 1013) Keyed string type support.

6.94.1 Detailed Description

<<*interface*>> (p. 1013) Keyed string type support.

6.95 DDS_LatencyBudgetQosPolicy Struct Reference

<<*spec*>> (p. 790) 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.

Data Fields

- struct **DDS_Duration_t** **duration**
<<*spec*>> (p. 790) *Duration of the maximum acceptable delay.*

6.95.1 Detailed Description

<<*spec*>> (p. 790) 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:

DDS_Topic (p. 175), **DDS_DataReader** (p. 615), **DDS_DataWriter** (p. 485)

Status:

DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 1229), **DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 1229)

Properties:

RxO (p. 1245) = YES

Changeable (p. 1245) = **YES** (p. 1245)

See also

DDS_PublishModeQosPolicy (p. 1920)

DDS_FlowController (p. 559)

6.95.2 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. 1319) **DDS_DataWriter** (p. 485) instances associated with a **DDS_EDF_FLOW_CONTROLLER_SCHED_POLICY** (p. 560) **DDS_FlowController** (p. 559) only. Together with the time of write, **DDS_LatencyBudgetQosPolicy::duration** (p. 1815) determines the deadline of each individual sample. RTI Connext uses this information to prioritize the sending of asynchronously published data; see **DDS_AsynchronousPublisherQosPolicy** (p. 1539).

6.95.3 Compatibility

The value offered is considered compatible with the value requested if and only if the inequality *offered duration* ≤ *requested duration* evaluates to 'TRUE'.

6.95.4 Field Documentation

6.95.4.1 duration

struct **DDS_Duration_t** DDS_LatencyBudgetQosPolicy::duration

<<*spec*>> (p. 790) Duration of the maximum acceptable delay.

[**default**] 0 (meaning minimize the delay)

Categories

Mutable (p. 949), **Discovery Metadata** (p. 958), **Matching** (p. 963), **Data Availability** (p. 843)

6.96 DDS_LifespanQosPolicy Struct Reference

<<*spec*>> (p. 790) Specifies how long the data written by the **DDS_DataWriter** (p. 485) is considered valid.

Data Fields

- struct **DDS_Duration_t** duration
 <<*basic*>> (p. 783) <<*spec*>> (p. 790) Maximum duration for the data's validity.

6.96.1 Detailed Description

<<*spec*>> (p. 790) Specifies how long the data written by the **DDS_DataWriter** (p. 485) is considered valid.

Each data sample written by the **DDS_DataWriter** (p. 485) 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 **DDS_DataReader** (p. 615) caches as well as from the transient and persistent information caches.

The expiration time of each sample from the **DDS_DataWriter** (p. 485)'s cache is computed by adding the duration specified by this QoS policy to the time when the sample is added to the **DDS_DataWriter** (p. 485)'s cache. This timestamp is not necessarily equal to the sample's source timestamp that can be provided by the user using the **FooDataWriter_write_w_timestamp** (p. 500) or **FooDataWriter_write_w_params** (p. 501) API.

The expiration time of each sample from the **DDS_DataReader** (p. 615)'s cache is computed by adding the duration to the reception timestamp.

See also

FooDataWriter_write (p. 496)

FooDataWriter_write_w_timestamp (p. 500)

Entity:

DDS_Topic (p. 175), **DDS_DataWriter** (p. 485)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **YES** (p. 1245)

6.96.2 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. 1805)), 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. 1989)

DDS_SampleInfo::reception_timestamp (p. 1992)

6.96.3 Field Documentation

6.96.3.1 duration

```
struct DDS_Duration_t DDS_LifespanQosPolicy::duration
```

<<**basic**>> (p. 783) <<**spec**>> (p. 790) Maximum duration for the data's validity.

[default] **DDS_DURATION_INFINITE** (p. 1208)

[range] [1 nanosec, 1 year] or **DDS_DURATION_INFINITE** (p. 1208)

Categories

Mutable (p. 949), **Discovery Metadata** (p. 958), **Durability** (p. 860), **Data Availability** (p. 843)

6.97 DDS_Listener Struct Reference

<<**interface**>> (p. 1013) Abstract base class for all Listener interfaces.

Data Fields

- void * **listener_data**

A place for listener implementors to keep a pointer to data that may be needed by their listener.

6.97.1 Detailed Description

<<*interface*>> (p. 1013) Abstract base class for all Listener interfaces.

Entity:

DDS_Entity (p. 1360)

QoS:

All QoS Policies (p. 1238)

Status:

Status Kinds (p. 1222)

All the supported kinds of concrete **DDS_Listener** (p. 1816) interfaces (one per concrete **DDS_Entity** (p. 1360) type) derive from this root and add functions whose prototype depends on the concrete Listener.

Listeners provide a way for RTI Connex 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 **DDS_Listener** (p. 1816) instance may be shared among multiple entities if you so desire. Consequently, the provided parameter contains a reference to the concerned **DDS_Entity** (p. 1360).

6.97.2 Access to Plain Communication Status

The general mapping between the plain communication statuses (see **Status Kinds** (p. 1222)) 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. 1222).
- `on_<communication_status>` is available on the relevant **DDS_Entity** (p. 1360) 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 Connex which operations are enabled within the listener (cf. operation **DDS_Entity** (p. 1360) `set_listener()`).
- When a plain communication status changes, RTI Connex 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 **DDS_Domain↵Participant** (p. 77)) and to set dedicated behaviors only where needed.

6.97.3 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 Connex tries to trigger the **DDS_SubscriberListener::on_data_on_readers** (p. 2010) with a parameter of the related **DDS_Subscriber** (p. 572);
- If this does not succeed (there is no listener or the operation is not enabled), RTI Connex tries to trigger **DDS_ReaderListener::on_data_available** (p. 1595) on all the related **DDS_DataReaderListener** (p. 1593) objects, with a parameter of the related **DDS_DataReader** (p. 615).

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 **DDS_DataReader** (p. 615) objects by calling **DDS_Subscriber_get_datareaders** (p. 588) on the related **DDS_Subscriber** (p. 572) and then get the data by calling **FooDataReader_read** (p. 625) or **FooDataReader_take** (p. 626) on the returned **DDS_DataReader** (p. 615) objects), or it wants to treat each **DDS_DataReader** (p. 615) independently and it may choose the second option (and then get the data by calling **FooDataReader_read** (p. 625) or **FooDataReader_take** (p. 626) on the related **DDS_DataReader** (p. 615)).

Note that if **DDS_SubscriberListener::on_data_on_readers** (p. 2010) is called, RTI Connex will *not* try to call **DDS_ReaderListener::on_data_available** (p. 1595). However, an application can force a call to the **DDS_DataReader** (p. 615) objects that have data by calling **DDS_Subscriber_notify_datareaders** (p. 590).

6.97.4 Operations Allowed in Listener Callbacks

See `Restricted Operations in Listener Callbacks`, in the Core Libraries User's Manual.

6.97.5 Best Practices with Listeners

Note that all the issues described below are avoided by using **DDS_WaitSet** (p. 1371).

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, Connex 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 **DDS_WaitSet** (p. 1371).

Avoid taking application mutexes/semaphores in Listener callbacks

Taking application mutexes/semaphores 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 **DDS_DataReaderListener::on_data_available()** (p. 1595) 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 **DDS_DataWriter_wait_for_acknowledgments** (p. 543) within the **DDS_DataReaderListener::on_data_available()** (p. 1595) callback. This will lead to deadlock.

See also

Status Kinds (p. 1222)

DDS_WaitSet (p. 1371), **DDS_Condition** (p. 1370)

6.97.6 Field Documentation

6.97.6.1 listener_data

```
void* DDS_Listener::listener_data
```

A place for listener implementors to keep a pointer to data that may be needed by their listener.

6.98 DDS_LivelinessChangedStatus Struct Reference

DDS_LIVELINESS_CHANGED_STATUS (p. 1231)

Data Fields

- **DDS_Long alive_count**

*The total count of currently alive **DDS_DataWriter** (p. 485) entities that write the **DDS_Topic** (p. 175) that this **DDS_↔_DataReader** (p. 615) reads.*

- **DDS_Long not_alive_count**

*The total count of currently not_alive **DDS_DataWriter** (p. 485) entities that write the **DDS_Topic** (p. 175) that this **DDS_↔_DataReader** (p. 615) 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 **DDS_DataWriter** (p. 485) was the last to cause this Data↔Reader's status to change, using **DDS_DataReader_get_matched_publication_data** (p. 679).*

6.98.1 Detailed Description

DDS_LIVELINESS_CHANGED_STATUS (p. 1231)

The **DDS_DataReaderListener::on_liveliness_changed** (p. 1594) callback may be invoked for the following reasons:

- The liveliness of any **DDS_DataWriter** (p. 485) matching this DataReader (as defined by the **DDS_Liveliness_↔QosPolicyKind** (p. 1296) 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. 1881)). In this case, RTI Connexx 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. 1821) will decrease (since there's one less matching alive DataWriter) and **DDS_Liveliness_↔ChangedStatus::not_alive_count** (p. 1821) 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. 1821) will remain the same (since the matching DataWriter was not alive) and **DDS_↔_LivelinessChangedStatus::not_alive_count** (p. 1821) 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 Data↔Writer has lost liveliness. For example, when the **DDS_LivelinessQosPolicyKind** (p. 1296) 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. 1881)) has changed such that a DataWriter that was unmatched with the DataReader now matches.

Examples

HelloWorld_subscriber.c.

6.98.2 Field Documentation

6.98.2.1 alive_count

DDS_Long DDS_LivelinessChangedStatus::alive_count

The total count of currently alive **DDS_DataWriter** (p. 485) entities that write the **DDS_Topic** (p. 175) that this **DDS_DataReader** (p. 615) reads.

6.98.2.2 not_alive_count

DDS_Long DDS_LivelinessChangedStatus::not_alive_count

The total count of currently not_alive **DDS_DataWriter** (p. 485) entities that write the **DDS_Topic** (p. 175) that this **DDS_DataReader** (p. 615) reads.

6.98.2.3 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.

6.98.2.4 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.

6.98.2.5 last_publication_handle

`DDS_InstanceHandle_t DDS_LivelinessChangedStatus::last_publication_handle`

This InstanceHandle can be used to look up which remote **DDS_DataWriter** (p. 485) was the last to cause this DataReader's status to change, using **DDS_DataReader_get_matched_publication_data** (p. 679).

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 **DDS_DataReader_get_matched_publication_data** (p. 679) 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.

6.99 DDS_LivelinessLostStatus Struct Reference

DDS_LIVELINESS_LOST_STATUS (p. 1231)

Data Fields

- **DDS_Long total_count**

*Total cumulative number of times that a previously-alive **DDS_DataWriter** (p. 485) became not alive due to a failure 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.

6.99.1 Detailed Description

DDS_LIVELINESS_LOST_STATUS (p. 1231)

Entity:

DDS_DataWriter (p. 485)

Listener:

DDS_DataWriterListener (p. 1644)

The liveliness that the **DDS_DataWriter** (p. 485) has committed through its **DDS_LivelinessQosPolicy** (p. 1823) was not respected; thus **DDS_DataReader** (p. 615) entities will consider the **DDS_DataWriter** (p. 485) as no longer "alive/active".

6.99.2 Field Documentation

6.99.2.1 total_count

`DDS_Long DDS_LivelinessLostStatus::total_count`

Total cumulative number of times that a previously-alive **DDS_DataWriter** (p. 485) became not alive due to a failure to actively signal its liveliness within the offered liveliness period.

This count does not change when an already not alive **DDS_DataWriter** (p. 485) simply remains not alive for another liveliness period.

6.99.2.2 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.

6.100 DDS_LivelinessQosPolicy Struct Reference

Specifies and configures the mechanism that allows **DDS_DataReader** (p. 615) entities to detect when **DDS_DataWriter** (p. 485) entities become disconnected or "dead."

Data Fields

- **DDS_LivelinessQosPolicyKind kind**
<<basic>> (p. 783) <<spec>> (p. 790) The kind of liveliness desired.
- struct **DDS_Duration_t lease_duration**
*<<basic>> (p. 783) <<spec>> (p. 790) The duration within which a **DDS_DataWriter** (p. 485) must be asserted, or else it is assumed to be not alive.*
- **DDS_Long assertions_per_lease_duration**
*<<basic>> (p. 783) <<extension>> (p. 795) The number of assertions a **DDS_DataWriter** (p. 485) will send during a its **DDS_LivelinessQosPolicy::lease_duration** (p. 1825).*

6.100.1 Detailed Description

Specifies and configures the mechanism that allows **DDS_DataReader** (p. 615) entities to detect when **DDS_DataWriter** (p. 485) entities become disconnected or "dead."

The liveliness status of a **DDS_DataWriter** (p. 485) is used to maintain instance ownership in combination with the setting of the **OWNERSHIP** (p. 1301) policy. The application is also informed via **DDS_Listener** (p. 1816) when an **DDS_DataWriter** (p. 485) is no longer alive.

A **DDS_DataWriter** (p. 485) commits to signalling its liveliness at intervals not to exceed the **DDS_LivelinessQosPolicy::lease_duration** (p. 1825) configured on the **DDS_DataWriter** (p. 485). The rate at which the **DDS_DataWriter** (p. 485) will signal its liveliness is defined by **DDS_LivelinessQosPolicy::assertions_per_lease_duration** (p. 1826).

The **DDS_DataReader** (p. 615) `lease_duration` specifies the maximum period at which matching DataWriters must have their liveliness asserted.

In addition, in the subscribing application Connex 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 **DDS_DataReader** (p. 615) of loss of liveliness and **DDS_DataWriter** (p. 485) 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 **DDS_DataWriter** (p. 485) 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:

DDS_Topic (p. 175), **DDS_DataReader** (p. 615), **DDS_DataWriter** (p. 485)

Status:

DDS_LIVELINESS_LOST_STATUS (p. 1231), **DDS_LivelinessLostStatus** (p. 1822);
DDS_LIVELINESS_CHANGED_STATUS (p. 1231), **DDS_LivelinessChangedStatus** (p. 1820);
DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS (p. 1229), **DDS_OFFERED_INCOMPATIBLE_QOS_↵**
STATUS (p. 1229)

Properties:

RxO (p. 1245) = YES
Changeable (p. 1245) = **UNTIL ENABLE** (p. 1245)

6.100.2 Usage

This policy controls the mechanism and parameters used by RTI Connex 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. 1301) 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. 1297) 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 Connex takes responsibility for renewing the leases at the required rates and thus, as long as the local process where a **DDS_↵**
DomainParticipant (p. 77) is running and the link connecting it to remote participants remains connected, the entities within the **DDS_DomainParticipant** (p. 77) will be considered alive. This requires the lowest overhead.

The manual settings (**DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS** (p. 1297), **DDS_MANUAL_BY_↵**
TOPIC_LIVELINESS_QOS (p. 1297)) require the application on the publishing side to periodically assert the liveliness before the lease expires to indicate the corresponding **DDS_Entity** (p. 1360) is still alive. The action can be explicit by calling the **DDS_DataWriter_assert_liveliness** (p. 538) 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. 1297) requires only that one **DDS_Entity** (p. 1360) within a participant is asserted to be alive to deduce all other **DDS_Entity** (p. 1360) objects within the same **DDS_DomainParticipant** (p. 77) are also alive.
- The setting **DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS** (p. 1297) requires that at least one instance within the **DDS_DataWriter** (p. 485) is asserted.

Changes in **LIVELINESS** (p. 1296) must be detected by the Service with a time-granularity greater or equal to the **DDS_LivelinessQosPolicy::lease_duration** (p. 1825). This ensures that the value of the **DDS_LivelinessChangedStatus** (p. 1820) is updated at least once during each lease_duration and the related Listeners and **DDS_WaitSet** (p. 1371) s are notified within a lease_duration from the time the **LIVELINESS** (p. 1296) changed.

6.100.3 Compatibility

The value offered is considered compatible with the value requested if and only if the following conditions are met:

- the inequality *offered kind* $\geq$ *requested kind* evaluates to 'TRUE'. For the purposes of this inequality, the values of **DDS_LivelinessQosPolicyKind** (p. 1296) kind are considered ordered such that: **DDS_AUTOMATIC_LIVELINESS_QOS** (p. 1297) < **DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS** (p. 1297) < **DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS** (p. 1297).
- the inequality *offered lease_duration* $\leq$ *requested lease_duration* evaluates to **DDS_BOOLEAN_TRUE** (p. 1201).

See also

Relationship between registration, liveliness and ownership (p. 1866)

6.100.4 Field Documentation

6.100.4.1 kind

DDS_LivelinessQosPolicyKind DDS_LivelinessQosPolicy::kind

<<**basic**>> (p. 783) <<**spec**>> (p. 790) The kind of liveliness desired.

[default] **DDS_AUTOMATIC_LIVELINESS_QOS** (p. 1297)

Categories

Immutable (p. 914), **Discovery Metadata** (p. 958), **Matching** (p. 963), **Liveliness** (p. 874)

6.100.4.2 lease_duration

```
struct DDS_Duration_t DDS_LivelinessQosPolicy::lease_duration
```

<<**basic**>> (p. 783) <<**spec**>> (p. 790) The duration within which a **DDS_DataWriter** (p. 485) 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.

[default] DDS_DURATION_INFINITE (p. 1208)

[range] [0,1 year] or DDS_DURATION_INFINITE (p. 1208)

Categories

Immutable (p. 914), **Discovery Metadata** (p. 958), **Matching** (p. 963), **Liveliness** (p. 874)

6.100.4.3 assertions_per_lease_duration

```
DDS_Long DDS_LivelinessQosPolicy::assertions_per_lease_duration
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) The number of assertions a **DDS_DataWriter** (p. 485) will send during a its **DDS_LivelinessQosPolicy::lease_duration** (p. 1825).

This field only applies to a **DDS_DataWriter** (p. 485) 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. 914), **Liveliness** (p. 874)

6.101 DDS_Locator_t Struct Reference

<<**extension**>> (p. 795) Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports.

Data Fields

- **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. 1104) octet field to hold the IP address.

6.101.1 Detailed Description

<<**extension**>> (p. 795) Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports.

6.101.2 Field Documentation

6.101.2.1 `kind`

DDS_Long `DDS_Locator_t::kind`

The kind of locator.

If the `Locator_t` kind is **DDS_LOCATOR_KIND_UDPv4** (p. 1110), the address contains an IPv4 address. In this case, the leading 12 octets of the **DDS_Locator_t::address** (p. 1827) must be zero. The last 4 octets of **DDS_Locator_t::address** (p. 1827) are used to store the IPv4 address.

If the `Locator_t` kind is **DDS_LOCATOR_KIND_UDPv6** (p. 1110), 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. 1827) field.

6.101.2.2 `port`

DDS_UnsignedLong `DDS_Locator_t::port`

the port number

6.101.2.3 `address`

DDS_Octet `DDS_Locator_t::address[ DDS_LOCATOR_ADDRESS_LENGTH_MAX]`

A **DDS_LOCATOR_ADDRESS_LENGTH_MAX** (p. 1104) octet field to hold the IP address.

6.102 DDS_LocatorFilter_t Struct Reference

<<**extension**>> (p. 795) Specifies the configuration of an individual channel within a MultiChannel DataWriter.

Data Fields

- struct **DDS_LocatorSeq locators**
 <<**extension**>> (p. 795) Sequence containing from one to 16 **DDS_Locator_t** (p. 1826), used to specify the multicast address locators of an individual channel within a MultiChannel DataWriter.
- char * **filter_expression**
 <<**extension**>> (p. 795) A logical expression used to determine the data that will be published in the channel.

6.102.1 Detailed Description

<<**extension**>> (p. 795) Specifies the configuration of an individual channel within a MultiChannel DataWriter.

QoS:

DDS_LocatorFilterQosPolicy (p. 1829)

6.102.2 Field Documentation

6.102.2.1 locators

```
struct DDS_LocatorSeq DDS_LocatorFilter_t::locators
```

<<**extension**>> (p. 795) Sequence containing from one to 16 **DDS_Locator_t** (p. 1826), used to specify the multicast address locators of an individual channel within a MultiChannel DataWriter.

[default] Empty sequence.

6.102.2.2 filter_expression

```
char* DDS_LocatorFilter_t::filter_expression
```

<<**extension**>> (p. 795) 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. 1829)

Important: This value must be an allocated string with **DDS_String_alloc** (p. 1528) or **DDS_String_dup** (p. 1528). It should not be assigned to a string constant.

See also

Queries and Filters Syntax (p. 738)

[default] NULL (invalid value)

6.103 DDS_LocatorFilterQosPolicy Struct Reference

<<**extension**>> (p. 795) The QoS policy used to report the configuration of a MultiChannel DataWriter as part of **DDS_PublicationBuiltinTopicData** (p. 1905).

Data Fields

- struct **DDS_LocatorFilterSeq** **locator_filters**
 <<**extension**>> (p. 795) A sequence of **DDS_LocatorFilter_t** (p. 1828). Each **DDS_LocatorFilter_t** (p. 1828) reports the configuration of a single channel of a MultiChannel DataWriter.
- char * **filter_name**
 <<**extension**>> (p. 795) Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.

6.103.1 Detailed Description

<<**extension**>> (p. 795) The QoS policy used to report the configuration of a MultiChannel DataWriter as part of **DDS_PublicationBuiltinTopicData** (p. 1905).

Entity:

DDS_PublicationBuiltinTopicData (p. 1905)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

6.103.2 Field Documentation

6.103.2.1 locator_filters

```
struct DDS_LocatorFilterSeq DDS_LocatorFilterQosPolicy::locator_filters
```

<<**extension**>> (p. 795) A sequence of **DDS_LocatorFilter_t** (p. 1828). Each **DDS_LocatorFilter_t** (p. 1828) reports the configuration of a single channel of a MultiChannel DataWriter.

A sequence length of zero indicates the **DDS_MultiChannelQosPolicy** (p. 1856) is not in use.

[default] Empty sequence.

6.103.2.2 filter_name

```
char* DDS_LocatorFilterQosPolicy::filter_name
```

<<**extension**>> (p. 795) 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. 163) and **DDS_STRINGMATCHFILTER_NAME** (p. 164).

Warning

This value must be assigned to either one of the predefined values (**DDS_SQLFILTER_NAME** (p. 163) or **DDS_STRINGMATCHFILTER_NAME** (p. 164)), or to an allocated string with **DDS_String_alloc** (p. 1528) or **DDS_String_dup** (p. 1528). It should not be assigned to a string constant.

[default] **DDS_STRINGMATCHFILTER_NAME** (p. 164)

6.104 DDS_LocatorFilterSeq Struct Reference

Declares IDL sequence< **DDS_LocatorFilter_t** (p. 1828) >.

6.104.1 Detailed Description

Declares IDL sequence< **DDS_LocatorFilter_t** (p. 1828) >.

A sequence of **DDS_LocatorFilter_t** (p. 1828) used to report the channels' properties. If the length of the sequence is zero, the **DDS_MultiChannelQosPolicy** (p. 1856) is not in use.

Instantiates:

<<**generic**>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_LocatorFilter_t (p. 1828)

6.105 DDS_LocatorSeq Struct Reference

Declares IDL sequence < **DDS_Locator_t** (p. 1826) >

6.105.1 Detailed Description

Declares IDL sequence `< DDS_Locator_t (p. 1826) >`

See also

`DDS_Locator_t` (p. 1826)

6.106 DDS_LoggingQosPolicy Struct Reference

`<<extension>>` (p. 795) Configures the RTI Connex logging facility.

Data Fields

- **NDDS_Config_LogVerbosity verbosity**
`<<basic>>` (p. 783) `<<extension>>` (p. 795) *The verbositys at which RTI Connex diagnostic information is logged.*
- **NDDS_Config_LogCategory category**
`<<basic>>` (p. 783) `<<extension>>` (p. 795) *Categories of logged messages.*
- **NDDS_Config_LogPrintFormat print_format**
`<<extension>>` (p. 795) *The format used to output RTI Connex diagnostic information.*
- **char * output_file**
`<<basic>>` (p. 783) `<<extension>>` (p. 795) *Specifies the file to which log messages will be redirected to.*
- **char * output_file_suffix**
`<<basic>>` (p. 783) `<<extension>>` (p. 795) *Sets the file suffix when logging to a set of files*
- **DDS_Long max_bytes_per_file**
`<<extension>>` (p. 795) *Specifies the maximum number of bytes a single file can contain*
- **DDS_Long max_files**
`<<extension>>` (p. 795) *Specifies the maximum number of files to create before overwriting the previous ones.*

6.106.1 Detailed Description

`<<extension>>` (p. 795) Configures the RTI Connex logging facility.

All the properties associated with RTI Connex 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:

DDS_DomainParticipantFactory (p. 32)

Properties:

RxO (p. 1245) = NO

Changeable (p. 1245) = **Changeable** (p. 1245)

6.106.2 Field Documentation

6.106.2.1 verbosity

NDDS_Config_LogVerbosity DDS_LoggingQosPolicy::verbosity

<<**basic**>> (p. 783) <<**extension**>> (p. 795) The verbositys at which RTI Connnext diagnostic information is logged.

[default] **NDDS_CONFIG_LOG_VERBOSITY_ERROR** (p. 1443)

Categories

Mutable (p. 949), **Logging** (p. 875), **Debuggability** (p. 847)

6.106.2.2 category

NDDS_Config_LogCategory DDS_LoggingQosPolicy::category

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Categories of logged messages.

[default] Logging will be enabled for all the categories.

Categories

Mutable (p. 949), **Logging** (p. 875), **Debuggability** (p. 847)

6.106.2.3 print_format

NDDS_Config_LogPrintFormat DDS_LoggingQosPolicy::print_format

<<**extension**>> (p. 795) The format used to output RTI Connnext diagnostic information.

[default] **NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT** (p. 1446).

Categories

Mutable (p. 949), **Logging** (p. 875), **Debuggability** (p. 847)

6.106.2.4 output_file

char* DDS_LoggingQosPolicy::output_file

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Specifies the file to which log messages will be redirected to.

If the value of output_file is set to NULL, log messages will sent to standard output.

If **DDS_LoggingQosPolicy::max_bytes_per_file** (p. 1833) is not **DDS_LENGTH_UNLIMITED** (p. 1326), this is used as the file name prefix for a set of numbered files.

Important: This value must be an allocated string with **DDS_String_alloc** (p. 1528) or **DDS_String_dup** (p. 1528). It should not be assigned to a string constant.

[default] NULL

See also

NDDS_Config_Logger_set_output_file_name (p. 1451)

NDDS_Config_Logger_set_output_file_set (p. 1451)

Categories

Mutable (p. 949), **Logging** (p. 875), **Debuggability** (p. 847)

6.106.2.5 output_file_suffix

char* DDS_LoggingQosPolicy::output_file_suffix

<<**basic**>> (p. 783) <<**extension**>> (p. 795) 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. 1326).

It specifies the suffix to use for the set of files used to redirect the logging output. The prefix is **DDS_LoggingQos_↵Policy::output_file** (p. 1832).

Important: This value must be an allocated string with **DDS_String_alloc** (p. 1528) or **DDS_String_dup** (p. 1528). It should not be assigned to a string constant.

[default] NULL

[default] No suffix

See also

NDDS_Config_Logger_set_output_file_set (p. 1451)

Categories

Mutable (p. 949), **Logging** (p. 875), **Debuggability** (p. 847)

6.106.2.6 max_bytes_per_file

DDS_Long DDS_LoggingQosPolicy::max_bytes_per_file

<<**extension**>> (p. 795) Specifies the maximum number of bytes a single file can contain

When this field is different than **DDS_LENGTH_UNLIMITED** (p. 1326), it enables logging to separate files as they reach this size.

[default] **DDS_LENGTH_UNLIMITED** (p. 1326) (a single file is used)

See also

NDDS_Config_Logger_set_output_file_set (p. 1451)

Categories

Mutable (p. 949), **Logging** (p. 875), **Debuggability** (p. 847), **Middleware Memory Management** (p. 878)

6.106.2.7 max_files

DDS_Long DDS_LoggingQosPolicy::max_files

<<**extension**>> (p. 795) Specifies the maximum number of files to create before overwriting the previous ones.

Note

This field only applies when idref_LoggingQosPolicy_max_bytes_per_file is different than **DDS_LENGTH_UNLIMITED** (p. 1326).

When this field is different than **DDS_LENGTH_UNLIMITED** (p. 1326), and the number of files reaches this number, future logging messages overwrite the previously created files.

[default] **DDS_LENGTH_UNLIMITED** (p. 1326) (files aren't overwritten)

See also

NDDS_Config_Logger_set_output_file_set (p. 1451)

Categories

Mutable (p. 949), **Logging** (p. 875), **Debuggability** (p. 847), **Middleware Memory Management** (p. 878)

6.107 DDS_LongDoubleSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_LongDouble** (p. 1204) >

6.107.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_LongDouble** (p. 1204) >

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_LongDouble (p. 1204)

FooSeq (p. 2113)

6.108 DDS_LongLongSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_LongLong** (p. 1203) >

6.108.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_LongLong** (p. 1203) >

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_LongLong (p. 1203)

FooSeq (p. 2113)

6.109 DDS_LongSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_Long** (p. 1203) >

6.109.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_Long** (p. 1203) >

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_Long (p. 1203)

FooSeq (p. 2113)

6.110 DDS_MonitoringDedicatedParticipantSettings Struct Reference

<<*extension*>> (p. 795) Configures the use of a dedicated **DDS_DomainParticipant** (p. 77) to distribute the RTI Connex application telemetry data.

Data Fields

- **DDS_Boolean enable**
 <<*basic*>> (p. 783) <<*extension*>> (p. 795) Enables the use of a dedicated **DDS_DomainParticipant** (p. 77) to distribute the RTI Connex application telemetry data.
- int **domain_id**
 <<*basic*>> (p. 783) <<*extension*>> (p. 795) The domain ID used in the creation of RTI Monitoring Library 2.0 **DDS_DomainParticipant** (p. 77).
- char * **participant_qos_profile_name**
 <<*extension*>> (p. 795) The fully qualified name of the profile used to configure the **DDS_DomainParticipant** (p. 77) that will be used to distribute telemetry data.
- struct **DDS_StringSeq collector_initial_peers**
 <<*basic*>> (p. 783) <<*extension*>> (p. 795) 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. 1839).

6.110.1 Detailed Description

<<*extension*>> (p. 795) Configures the use of a dedicated **DDS_DomainParticipant** (p. 77) to distribute the RTI Connex application telemetry data.

6.110.2 Field Documentation

6.110.2.1 enable

`DDS_Boolean DDS_MonitoringDedicatedParticipantSettings::enable`

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Enables the use of a dedicated **DDS_DomainParticipant** (p. 77) to distribute the RTI Connex application telemetry data.

Setting this value to **DDS_BOOLEAN_FALSE** (p. 1201) is not currently supported.

[default] **DDS_BOOLEAN_TRUE** (p. 1201)

Categories

Mutable (p. 949), **Debuggability** (p. 847), **QoS Profile Management** (p. 893)

6.110.2.2 domain_id

`int DDS_MonitoringDedicatedParticipantSettings::domain_id`

<<**basic**>> (p. 783) <<**extension**>> (p. 795) The domain ID used in the creation of RTI Monitoring Library 2.0 **DDS_DomainParticipant** (p. 77).

[default] 2

Categories

Mutable (p. 949), **Debuggability** (p. 847), **Discovery** (p. 853), **Discovery Isolation** (p. 858)

6.110.2.3 participant_qos_profile_name

`char* DDS_MonitoringDedicatedParticipantSettings::participant_qos_profile_name`

<<**extension**>> (p. 795) The fully qualified name of the profile used to configure the **DDS_DomainParticipant** (p. 77) 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. 1410).

[default] NULL

Categories

Mutable (p. 949), **Debuggability** (p. 847), **QoS Profile Management** (p. 893)

6.110.2.4 collector_initial_peers

```
struct DDS_StringSeq DDS_MonitoringDedicatedParticipantSettings::collector_initial_peers
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) 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. 1839).

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 **DDS_DomainParticipant** (p. 77), which usually has 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. 1716) list of **DDS_MonitoringDedicatedParticipantSettings::participant_qos_profile_name** (p. 1837) will be used as the initial peers of **DDS_MonitoringDistributionSettings::dedicated_participant** (p. 1839).

[default] An empty sequence.

See also

DDS_DiscoveryQosPolicy::initial_peers (p. 1716) for further information about initial peers.

Categories

Mutable (p. 949), **Debuggability** (p. 847), **Discovery** (p. 853), **Discovery Isolation** (p. 858)

6.111 DDS_MonitoringDistributionSettings Struct Reference

<<**extension**>> (p. 795) Configures the distribution of telemetry data.

Data Fields

- struct **DDS_MonitoringDedicatedParticipantSettings** **dedicated_participant**
 - <<**extension**>> (p. 795) Configures the use of a dedicated **DDS_DomainParticipant** (p. 77) to distribute the RTI Connext application telemetry data.
- char * **publisher_qos_profile_name**
 - <<**extension**>> (p. 795) The fully qualified name of the profile used to configure the Publishers that distribute telemetry data.
- struct **DDS_MonitoringEventDistributionSettings** **event_settings**
 - <<**extension**>> (p. 795) Configures the distribution of event metrics.
- struct **DDS_MonitoringPeriodicDistributionSettings** **periodic_settings**
 - <<**extension**>> (p. 795) Configures the distribution of periodic metrics.
- struct **DDS_MonitoringLoggingDistributionSettings** **logging_settings**
 - <<**extension**>> (p. 795) Configures the distribution of logging messages.

6.111.1 Detailed Description

<<*extension*>> (p. 795) Configures the distribution of telemetry data.

6.111.2 Field Documentation

6.111.2.1 dedicated_participant

```
struct DDS_MonitoringDedicatedParticipantSettings DDS_MonitoringDistributionSettings::dedicated←  
_participant
```

<<*extension*>> (p. 795) Configures the use of a dedicated **DDS_DomainParticipant** (p. 77) to distribute the RTI Connex application telemetry data.

Categories

Mutable (p. 949), **Debuggability** (p. 847), **QoS Profile Management** (p. 893)

6.111.2.2 publisher_qos_profile_name

```
char* DDS_MonitoringDistributionSettings::publisher_qos_profile_name
```

<<*extension*>> (p. 795) The fully qualified name of the profile used to configure the Publishers that distribute telemetry data.

There is one Publisher for each telemetry data **DDS_Topic** (p. 175): **RTI_MONITORING_PERIODIC_TOPIC_NAME** (p. 1436), **RTI_MONITORING_EVENT_TOPIC_NAME** (p. 1436), and **RTI_MONITORING_LOGGING_TOPIC_NAME** (p. 1436).

If NULL (the default value) then RTI Monitoring Library 2.0 uses **DDS_PROFILE_GENERIC_MONITORING2** (p. 1410).

[default] NULL

Categories

Mutable (p. 949), **Debuggability** (p. 847), **QoS Profile Management** (p. 893)

6.111.2.3 event_settings

```
struct DDS_MonitoringEventDistributionSettings DDS_MonitoringDistributionSettings::event_settings
```

<<**extension**>> (p. 795) Configures the distribution of event metrics.

Categories

Mutable (p. 949), **Debuggability** (p. 847)

6.111.2.4 periodic_settings

```
struct DDS_MonitoringPeriodicDistributionSettings DDS_MonitoringDistributionSettings::periodic_↵  
settings
```

<<**extension**>> (p. 795) Configures the distribution of periodic metrics.

Categories

Mutable (p. 949), **Debuggability** (p. 847)

6.111.2.5 logging_settings

```
struct DDS_MonitoringLoggingDistributionSettings DDS_MonitoringDistributionSettings::logging_↵  
settings
```

<<**extension**>> (p. 795) Configures the distribution of logging messages.

Categories

Mutable (p. 949), **Debuggability** (p. 847), **Logging** (p. 875)

6.112 DDS_MonitoringEventDistributionSettings Struct Reference

<<**extension**>> (p. 795) Configures the distribution of event metrics.

Data Fields

- **DDS_UnsignedLong concurrency_level**
 <<extension>> (p. 795) Defines how concurrent the push of event metrics to RTI Monitoring Library 2.0 is.
- char * **datawriter_qos_profile_name**
 <<extension>> (p. 795) The fully qualified name of the profile used to configure the **DDS_DataWriter** (p. 485) that distributes event metrics.
- struct **DDS_ThreadSettings_t thread**
 <<extension>> (p. 795) The settings of the event metric thread.
- struct **DDS_Duration_t publication_period**
 <<basic>> (p. 783) <<extension>> (p. 795) Period at which the event metric thread publishes the event metrics that have changed since the last time they were published.

6.112.1 Detailed Description

<<extension>> (p. 795) 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 **DDS_DataWriter** (p. 485) is lost, a matching **DDS_DataReader** (p. 615) will push the new value of **DDS_LivelinessChangedStatus** (p. 1820) 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. 1820).
- **Resource metrics:** Provided to RTI Monitoring Library 2.0 when a resource (such as **DDS_DataWriter** (p. 485)) is created or deleted.

The event metrics that will be distributed for an observable resource can be configured with **DDS_Monitoring_↔_TelemetryData::metrics** (p. 1855).

6.112.2 Field Documentation

6.112.2.1 concurrency_level

`DDS_UnsignedLong DDS_MonitoringEventDistributionSettings::concurrency_level`

<<**extension**>> (p. 795) 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 **DDS_DataReader** (p. 615)) 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. 1842).

[default] 5

[range] [1, 100]

Categories

Mutable (p. 949), **Concurrency** (p. 839), **Debuggability** (p. 847)

6.112.2.2 datawriter_qos_profile_name

`char* DDS_MonitoringEventDistributionSettings::datawriter_qos_profile_name`

<<**extension**>> (p. 795) The fully qualified name of the profile used to configure the **DDS_DataWriter** (p. 485) that distributes event metrics.

The **DDS_DataWriter** (p. 485) Topic is **RTI_MONITORING_EVENT_TOPIC_NAME** (p. 1436).

If NULL (the default value), then RTI Monitoring Library 2.0 uses **DDS_PROFILE_GENERIC_MONITORING2** (p. 1410).

[default] NULL

Categories

Mutable (p. 949), **Debuggability** (p. 847), **QoS Profile Management** (p. 893)

6.112.2.3 thread

```
struct DDS_ThreadSettings_t DDS_MonitoringEventDistributionSettings::thread
```

<<**extension**>> (p. 795) 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. 1843).

[default] DDS_THREAD_SETTINGS_DEFAULT

Categories

Mutable (p. 949), **Debuggability** (p. 847), **Concurrency** (p. 839)

6.112.2.4 publication_period

```
struct DDS_Duration_t DDS_MonitoringEventDistributionSettings::publication_period
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) 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. 949), **Debuggability** (p. 847), **Data Availability** (p. 843)

6.113 DDS_MonitoringLoggingDistributionSettings Struct Reference

<<**extension**>> (p. 795) Configures the distribution of log messages.

Data Fields

- **DDS_UnsignedLong concurrency_level**
 <<*extension*>> (p. 795) *Defines how concurrent the push of log messages to RTI Monitoring Library 2.0 is.*
- **DDS_UnsignedLong max_historical_logs**
 <<*basic*>> (p. 783) <<*extension*>> (p. 795) *The number of log messages that RTI Monitoring Library 2.0 will keep as history.*
- **char * datawriter_qos_profile_name**
 <<*extension*>> (p. 795) *The fully qualified name of the profile used to configure the **DDS_DataWriter** (p. 485) that distributes log messages.*
- **struct DDS_ThreadSettings_t thread**
 <<*extension*>> (p. 795) *The settings of the logging thread.*
- **struct DDS_Duration_t publication_period**
 <<*basic*>> (p. 783) <<*extension*>> (p. 795) *Period at which the logging thread publishes log messages.*

6.113.1 Detailed Description

<<*extension*>> (p. 795) 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. 1446) associated with the log message.

The default value of the forwarding level for all facilities is **NDDS_CONFIG_SYSLOG_VERBOSITY_WARNING** (p. 1448). This value can be changed with the **DDS_MonitoringTelemetryData::logs** (p. 1856) 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. 1845)).

6.113.2 Field Documentation

6.113.2.1 concurrency_level

DDS_UnsignedLong DDS_MonitoringLoggingDistributionSettings::concurrency_level

<<**extension**>> (p. 795) 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. 1846).

[default] 5

[range] [1, 100]

Categories

Mutable (p. 949), **Debuggability** (p. 847), **Logging** (p. 875), **Concurrency** (p. 839)

6.113.2.2 max_historical_logs

DDS_UnsignedLong DDS_MonitoringLoggingDistributionSettings::max_historical_logs

<<**basic**>> (p. 783) <<**extension**>> (p. 795) 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. 949), **Debuggability** (p. 847), **Logging** (p. 875), **Middleware Memory Management** (p. 878)

6.113.2.3 datawriter_qos_profile_name

```
char* DDS_MonitoringLoggingDistributionSettings::datawriter_qos_profile_name
```

<<**extension**>> (p. 795) The fully qualified name of the profile used to configure the **DDS_DataWriter** (p. 485) that distributes log messages.

The **DDS_DataWriter** (p. 485) Topic is **RTI_MONITORING_LOGGING_TOPIC_NAME** (p. 1436).

If NULL (the default value), then RTI Monitoring Library 2.0 uses **DDS_PROFILE_GENERIC_MONITORING2** (p. 1410).

[default] NULL

Categories

Mutable (p. 949), **Debuggability** (p. 847), **QoS Profile Management** (p. 893), **Logging** (p. 875)

6.113.2.4 thread

```
struct DDS_ThreadSettings_t DDS_MonitoringLoggingDistributionSettings::thread
```

<<**extension**>> (p. 795) 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. 1846).

[default] DDS_THREAD_SETTINGS_DEFAULT

Categories

Mutable (p. 949), **Debuggability** (p. 847), **Concurrency** (p. 839), **Logging** (p. 875)

6.113.2.5 publication_period

```
struct DDS_Duration_t DDS_MonitoringLoggingDistributionSettings::publication_period
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) 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. 949), **Debuggability** (p. 847), **Data Availability** (p. 843), **Logging** (p. 875)

6.114 DDS_MonitoringLoggingForwardingSettings Struct Reference

<<**extension**>> (p. 795) Configures the forwarding levels of log messages for the different **NDDS_Config_Log**↔
Facility (p. 1446).

Data Fields

- **NDDS_Config_SyslogVerbosity middleware_forwarding_level**
<<**basic**>> (p. 783) <<**extension**>> (p. 795) Log messages with **NDDS_CONFIG_LOG_FACILITY_MIDDLEWARE** (p. 1447) 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. 783) <<**extension**>> (p. 795) Log messages with **NDDS_CONFIG_LOG_FACILITY_SECURITY**↔
EVENT (p. 1447) 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. 783) <<**extension**>> (p. 795) Log messages with **NDDS_CONFIG_LOG_FACILITY_SERVICE** (p. 1447) 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. 783) <<**extension**>> (p. 795) Log messages with **NDDS_CONFIG_LOG_FACILITY_USER** (p. 1447) and a log level more verbose than this value will not be distributed by RTI Monitoring Library 2.0.

6.114.1 Detailed Description

<<**extension**>> (p. 795) Configures the forwarding levels of log messages for the different **NDDS_Config_Log**↔
Facility (p. 1446).

6.114.2 Field Documentation

6.114.2.1 middleware_forwarding_level

NDDS_Config_SyslogVerbosity DDS_MonitoringLoggingForwardingSettings::middleware_forwarding_level

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Log messages with **NDDS_CONFIG_LOG_FACILITY**↔
MIDDLEWARE (p. 1447) 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. 1448)

Keep in mind that setting this property beyond WARNING if your application's verbosity is greater than **NDDS**↔
CONFIG_LOG_VERBOSITY_WARNING (p. 1443), may affect performance due to the large amount of messages produced.

Categories

Mutable (p. 949), **Debuggability** (p. 847), **Logging** (p. 875)

6.114.2.2 security_event_forwarding_level

NDDS_Config_SyslogVerbosity DDS_MonitoringLoggingForwardingSettings::security_event_forwarding_level

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Log messages with **NDDS_CONFIG_LOG_FACILITY_SECURITY_EVENT** (p. 1447) 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. 1448)

Keep in mind that setting this property beyond WARNING if your application's verbosity is greater than **NDDS_CONFIG_LOG_VERBOSITY_WARNING** (p. 1443), may affect performance due to the large amount of messages produced.

Categories

Mutable (p. 949), **Debuggability** (p. 847), **Logging** (p. 875), **Security** (p. 906)

6.114.2.3 service_forwarding_level

NDDS_Config_SyslogVerbosity DDS_MonitoringLoggingForwardingSettings::service_forwarding_level

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Log messages with **NDDS_CONFIG_LOG_FACILITY_SERVICE** (p. 1447) 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. 1448)

Keep in mind that setting this property beyond WARNING if your application's verbosity is greater than **NDDS_CONFIG_LOG_VERBOSITY_WARNING** (p. 1443), may affect performance due to the large amount of messages produced.

Categories

Mutable (p. 949), **Debuggability** (p. 847), **Logging** (p. 875)

6.114.2.4 user_forwarding_level

NDDS_Config_SyslogVerbosity DDS_MonitoringLoggingForwardingSettings::user_forwarding_level

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Log messages with **NDDS_CONFIG_LOG_FACILITY_USER** (p. 1447) 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. 1448)

Keep in mind that setting this property beyond WARNING if your application's verbosity is greater than **NDDS_CONFIG_LOG_VERBOSITY_WARNING** (p. 1443), may affect performance due to the large amount of messages produced.

Categories

Mutable (p. 949), **Debuggability** (p. 847), **Logging** (p. 875)

6.115 DDS_MonitoringMetricSelection Struct Reference

<<**extension**>> (p. 795) This data structure is used to configure event and periodic metrics collection and distribution for a specific set of observable resources.

Data Fields

- char * **resource_selection**

<<**extension**>> (p. 795) An expression pattern that selects a subset of resources by matching the pattern to the resource names. **DDS_MonitoringMetricSelection::enabled_metrics_selection** (p. 1850) and **DDS_MonitoringMetricSelection::disabled_metrics_selection** (p. 1850) are applied to this subset of resources.

- struct **DDS_StringSeq enabled_metrics_selection**

<<**extension**>> (p. 795) 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. 1849).

- struct **DDS_StringSeq disabled_metrics_selection**

<<**extension**>> (p. 795) 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. 1849).

6.115.1 Detailed Description

<<**extension**>> (p. 795) This data structure is used to configure event and periodic metrics collection and distribution for a specific set of observable resources.

6.115.2 Field Documentation

6.115.2.1 resource_selection

```
char* DDS_MonitoringMetricSelection::resource_selection
```

<<**extension**>> (p. 795) An expression pattern that selects a subset of resources by matching the pattern to the resource names. **DDS_MonitoringMetricSelection::enabled_metrics_selection** (p. 1850) and **DDS_MonitoringMetricSelection::disabled_metrics_selection** (p. 1850) 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 hierarchy.

See the Telemetry Data / Resources chapter of the RTI Connex Observability Framework documentation for further information on the observable resource names and expression patterns.

Categories

Mutable (p. 949), **Debuggability** (p. 847)

6.115.2.2 enabled_metrics_selection

```
struct DDS_StringSeq DDS_MonitoringMetricSelection::enabled_metrics_selection
```

<<**extension**>> (p. 795) 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. 1849).

This sequence is evaluated first, followed by **DDS_MonitoringMetricSelection::disabled_metrics_selection** (p. 1850). 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. 1762)). The second pattern refers to all the DataReader **DDS_ReliabilityQosPolicy** (p. 1936) metrics. The last pattern selects all the available application metrics.

See the Telemetry Data / Metrics chapter of the RTI Connex Observability Framework documentation for further information on the metric names and metric patterns.

Categories

Mutable (p. 949), **Debuggability** (p. 847)

6.115.2.3 disabled_metrics_selection

```
struct DDS_StringSeq DDS_MonitoringMetricSelection::disabled_metrics_selection
```

<<**extension**>> (p. 795) 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. 1849).

This sequence is evaluated after **DDS_MonitoringMetricSelection::enabled_metrics_selection** (p. 1850). 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. 1762)). The second pattern refers to all the DataReader **DDS_ReliabilityQosPolicy** (p. 1936) metrics. The last pattern selects all the available application metrics.

See the Telemetry Data / Metrics chapter of the RTI Connex Observability Framework documentation for further information on the metric names and metric patterns.

Categories

Mutable (p. 949), **Debuggability** (p. 847)

6.116 DDS_MonitoringMetricSelectionSeq Struct Reference

Declares IDL sequence < **DDS_MonitoringMetricSelection** (p. 1849) >

6.116.1 Detailed Description

Declares IDL sequence < **DDS_MonitoringMetricSelection** (p. 1849) >

Instantiates:

<<**generic**>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_MonitoringMetricSelection (p. 1849)

6.117 DDS_MonitoringPeriodicDistributionSettings Struct Reference

<<**extension**>> (p. 795) Configures the distribution of periodic metrics.

Data Fields

- char * **datawriter_qos_profile_name**
<<**extension**>> (p. 795) *The fully qualified name of the profile used to configure the **DDS_DataWriter** (p. 485) that distributes periodic metrics.*
- struct **DDS_ThreadSettings_t** **thread**
<<**extension**>> (p. 795) *The settings of the periodic metric thread.*
- struct **DDS_Duration_t** **polling_period**
<<**basic**>> (p. 783) <<**extension**>> (p. 795) *Period at which the periodic metric thread polls and publishes the periodic metrics.*

6.117.1 Detailed Description

<<**extension**>> (p. 795) 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. 1654).

The periodic metrics that will be distributed for an observable resource can be configured with **DDS_Monitoring_↔TelemetryData::metrics** (p. 1855).

6.117.2 Field Documentation

6.117.2.1 datawriter_qos_profile_name

```
char* DDS_MonitoringPeriodicDistributionSettings::datawriter_qos_profile_name
```

<<**extension**>> (p. 795) The fully qualified name of the profile used to configure the **DDS_DataWriter** (p. 485) that distributes periodic metrics.

The **DDS_DataWriter** (p. 485) Topic is **RTI_MONITORING_PERIODIC_TOPIC_NAME** (p. 1436).

If NULL (the default value), then RTI Monitoring Library 2.0 uses **DDS_PROFILE_GENERIC_MONITORING2** (p. 1410).

[default] NULL

Categories

Mutable (p. 949), **Debuggability** (p. 847), **QoS Profile Management** (p. 893)

6.117.2.2 thread

```
struct DDS_ThreadSettings_t DDS_MonitoringPeriodicDistributionSettings::thread
```

<<**extension**>> (p. 795) 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. 1853).

[default] DDS_THREAD_SETTINGS_DEFAULT

Categories

Mutable (p. 949), **Debuggability** (p. 847), **Concurrency** (p. 839)

6.117.2.3 polling_period

```
struct DDS_Duration_t DDS_MonitoringPeriodicDistributionSettings::polling_period
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Period at which the periodic metric thread polls and publishes the periodic metrics.

[default] 5 seconds

[range] > 0 seconds

Categories

Mutable (p. 949), **Debuggability** (p. 847), **Data Availability** (p. 843)

6.118 DDS_MonitoringQosPolicy Struct Reference

<<**extension**>> (p. 795) Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connexx telemetry data.

Data Fields

- **DDS_Boolean enable**
 - <<**basic**>> (p. 783) <<**extension**>> (p. 795) *Enables the collection and distribution of telemetry data for an RTI Connexx application using RTI Monitoring Library 2.0.*
- **char * application_name**
 - <<**basic**>> (p. 783) <<**extension**>> (p. 795) *The name of the resource that represents this RTI Connexx application.*
- **struct DDS_MonitoringDistributionSettings distribution_settings**
 - <<**extension**>> (p. 795) *Configures the distribution of telemetry data.*
- **struct DDS_MonitoringTelemetryData telemetry_data**
 - <<**extension**>> (p. 795) *Configures the telemetry data that will be distributed.*

6.118.1 Detailed Description

<<**extension**>> (p. 795) Configures the use of RTI Monitoring Library 2.0 to collect and distribute RTI Connexx telemetry data.

6.118.2 Field Documentation

6.118.2.1 enable

DDS_Boolean DDS_MonitoringQosPolicy::enable

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Enables the collection and distribution of telemetry data for an RTI Connexx application using RTI Monitoring Library 2.0.

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. In that case, children entities from that entity (invisible to the library) will not be monitored either.

[default] DDS_BOOLEAN_FALSE (p. 1201)

Categories

Mutable (p. 949), **Debuggability** (p. 847)

6.118.2.2 application_name

char* DDS_MonitoringQosPolicy::application_name

<<**basic**>> (p. 783) <<**extension**>> (p. 795) The name of the resource that represents this RTI Connexx 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 Connexx 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. 949), **Debuggability** (p. 847)

6.118.2.3 distribution_settings

```
struct DDS_MonitoringDistributionSettings DDS_MonitoringQosPolicy::distribution_settings
```

<<**extension**>> (p. 795) Configures the distribution of telemetry data.

Categories

Mutable (p. 949), **Debuggability** (p. 847)

6.118.2.4 telemetry_data

```
struct DDS_MonitoringTelemetryData DDS_MonitoringQosPolicy::telemetry_data
```

<<**extension**>> (p. 795) Configures the telemetry data that will be distributed.

Categories

Mutable (p. 949), **Debuggability** (p. 847)

6.119 DDS_MonitoringTelemetryData Struct Reference

<<**extension**>> (p. 795) Configures the telemetry data that will be distributed.

Data Fields

- struct **DDS_MonitoringMetricSelectionSeq metrics**
 <<**basic**>> (p. 783) <<**extension**>> (p. 795) Sequence of **DDS_MonitoringMetricSelection** (p. 1849) 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. 783) <<**extension**>> (p. 795) **DDS_MonitoringLoggingForwardingSettings** (p. 1847) containing the **NDDS_Config_SyslogVerbosity** (p. 1447) levels that will be forwarded for the different **NDDS_Config_LogFacility** (p. 1446).

6.119.1 Detailed Description

<<**extension**>> (p. 795) Configures the telemetry data that will be distributed.

6.119.2 Field Documentation

6.119.2.1 metrics

```
struct DDS_MonitoringMetricSelectionSeq DDS_MonitoringTelemetryData::metrics
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Sequence of **DDS_MonitoringMetricSelection** (p. 1849) containing the event and periodic metrics that will be collected and distributed for a given set of observable resources.

The different **DDS_MonitoringMetricSelection** (p. 1849) 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. 1840) and **DDS_MonitoringPeriodicDistributionSettings** (p. 1852) for further information on how RTI Monitoring Library 2.0 distributes **metrics** (p. 1855).

Categories

Mutable (p. 949), **Debuggability** (p. 847)

6.119.2.2 logs

```
struct DDS_MonitoringLoggingForwardingSettings DDS_MonitoringTelemetryData::logs
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) **DDS_MonitoringLoggingForwardingSettings** (p. 1847) containing the **NDDS_Config_SyslogVerbosity** (p. 1447) levels that will be forwarded for the different **NDDS_Config_LogFacility** (p. 1446).

See also

DDS_MonitoringLoggingDistributionSettings (p. 1843) for further information on how RTI Monitoring Library 2.0 distributes log messages.

Categories

Mutable (p. 949), **Debuggability** (p. 847), **Logging** (p. 875)

6.120 DDS_MultiChannelQosPolicy Struct Reference

<<**extension**>> (p. 795) Configures the ability of a DataWriter to send data on different multicast groups (addresses) based on the value of the data.

Data Fields

- struct **DDS_ChannelSettingsSeq** **channels**
 <<*extension*>> (p. 795) A sequence of **DDS_ChannelSettings_t** (p. 1564) 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. 795) Name of the filter class used to describe the filter expressions of a MultiChannel DataWriter.

6.120.1 Detailed Description

<<*extension*>> (p. 795) 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 **DDS_DataWriter** (p. 485) across multiple channels. A *channel* is defined by a filter expression and a sequence of multicast locators.

Entity:

DDS_DataWriter (p. 485)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

6.120.2 Usage

By using this QoS, a **DDS_DataWriter** (p. 485) 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. 496), 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.

6.120.3 Field Documentation

6.120.3.1 channels

```
struct DDS_ChannelSettingsSeq DDS_MultiChannelQosPolicy::channels
```

<<**extension**>> (p. 795) A sequence of **DDS_ChannelSettings_t** (p. 1564) 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. 1856) is not in use.

The sequence length cannot be greater than **DDS_DomainParticipantResourceLimitsQosPolicy::channel_seq_max_length** (p. 1752).

[default] Empty sequence.

Categories

Immutable (p. 914), **Discovery Metadata** (p. 958), **Data Availability** (p. 843)

6.120.3.2 filter_name

```
char* DDS_MultiChannelQosPolicy::filter_name
```

<<**extension**>> (p. 795) 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. 163)
- **DDS_STRINGMATCHFILTER_NAME** (p. 164)

Warning

The value for this field can be one of the constants above or a string allocated with **DDS_String_dup()** (p. 1528) to specify your own filter. You should not assign a string literal. When you assign a new value with **DDS_String_dup()** (p. 1528), 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. 1528) are released when `writer_qos` is finalized.

More information about string handling can be found under **String Conventions** (p. 1527).

[default] DDS_STRINGMATCHFILTER_NAME (p. 164)

Categories

Immutable (p. 914), **Discovery Metadata** (p. 958), **Data Availability** (p. 843), **Content Filtering** (p. 841)

6.121 DDS_Octets Struct Reference

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

Data Fields

- **int length**
Number of octets to serialize.
- **unsigned char * value**
DDS_Octets (p. 1859) array value.

6.121.1 Detailed Description

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

6.122 DDS_OctetSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_Octet** (p. 1202) >

6.122.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_Octet** (p. 1202) >

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_Octet (p. 1202)

FooSeq (p. 2113)

6.123 DDS_OctetsSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_Octets** (p. 1859) > .

6.123.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_Octets** (p. 1859) > .

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_Octets (p. 1859)

6.124 DDS_OctetsTypeSupport Struct Reference

<<*interface*>> (p. 1013) **DDS_Octets** (p. 1859) type support.

6.124.1 Detailed Description

<<*interface*>> (p. 1013) DDS_Octets (p. 1859) type support.

6.125 DDS_OfferedDeadlineMissedStatus Struct Reference

DDS_OFFERED_DEADLINE_MISSED_STATUS (p. 1228)

Data Fields

- **DDS_Long total_count**
Total cumulative count of the number of times the DDS_DataWriter (p. 485) 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 DDS_DataWriter (p. 485) for which an offered deadline was missed.

6.125.1 Detailed Description

DDS_OFFERED_DEADLINE_MISSED_STATUS (p. 1228)

Entity:

DDS_DataWriter (p. 485)

Listener:

DDS_DataWriterListener (p. 1644)

The deadline that the **DDS_DataWriter** (p. 485) has committed through its **DDS_DeadlineQosPolicy** (p. 1684) was not respected for a specific instance.

6.125.2 Field Documentation

6.125.2.1 total_count

DDS_Long DDS_OfferedDeadlineMissedStatus::total_count

Total cumulative count of the number of times the **DDS_DataWriter** (p. 485) failed to write within its offered deadline.

Missed deadlines accumulate; that is, each deadline period the `total_count` will be incremented by one.

6.125.2.2 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.

6.125.2.3 last_instance_handle

DDS_InstanceHandle_t DDS_OfferedDeadlineMissedStatus::last_instance_handle

Handle to the last instance in the **DDS_DataWriter** (p. 485) for which an offered deadline was missed.

6.126 DDS_OfferedIncompatibleQosStatus Struct Reference

DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 1229)

Data Fields

- **DDS_Long total_count**

*Total cumulative number of times the concerned **DDS_DataWriter** (p. 485) discovered a **DDS_DataReader** (p. 615) for the same **DDS_Topic** (p. 175), common partition with a requested QoS that is incompatible with that offered by the **DDS_DataWriter** (p. 485).*

- **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. 1246) 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 **DDS_DataWriter** (p. 485) discovered a **DDS_DataReader** (p. 615) for the same **DDS_Topic** (p. 175) and common partition with a requested QoS that is incompatible with that offered by the **DDS_DataWriter** (p. 485).*

6.126.1 Detailed Description

DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 1229)

Entity:

DDS_DataWriter (p. 485)

Listener:

DDS_DataWriterListener (p. 1644)

The qos policy value was incompatible with what was requested.

6.126.2 Field Documentation

6.126.2.1 total_count

DDS_Long DDS_OfferedIncompatibleQosStatus::total_count

Total cumulative number of times the concerned **DDS_DataWriter** (p. 485) discovered a **DDS_DataReader** (p. 615) for the same **DDS_Topic** (p. 175), common partition with a requested QoS that is incompatible with that offered by the **DDS_DataWriter** (p. 485).

6.126.2.2 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.

6.126.2.3 last_policy_id

DDS_QosPolicyId_t DDS_OfferedIncompatibleQosStatus::last_policy_id

The **DDS_QosPolicyId_t** (p. 1246) of one of the policies that was found to be incompatible the last time an incompatibility was detected.

6.126.2.4 policies

struct DDS_QosPolicyCountSeq DDS_OfferedIncompatibleQosStatus::policies

A list containing for each policy the total number of times that the concerned **DDS_DataWriter** (p. 485) discovered a **DDS_DataReader** (p. 615) for the same **DDS_Topic** (p. 175) and common partition with a requested QoS that is incompatible with that offered by the **DDS_DataWriter** (p. 485).

6.127 DDS_OwnershipQosPolicy Struct Reference

<<spec>> (p. 790) Specifies whether it is allowed for multiple **DDS_DataWriter** (p. 485) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.

Data Fields

- **DDS_OwnershipQosPolicyKind** *kind*
 <<*basic*>> (p. 783) <<*spec*>> (p. 790) *The kind of ownership*

6.127.1 Detailed Description

<<*spec*>> (p. 790) Specifies whether it is allowed for multiple **DDS_DataWriter** (p. 485) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.

Entity:

DDS_Topic (p. 175), **DDS_DataReader** (p. 615), **DDS_DataWriter** (p. 485)

Status:

DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 1229), **DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS** (p. 1229)

Properties:

RxO (p. 1245) = YES
Changeable (p. 1245) = **UNTIL ENABLE** (p. 1245)

See also

OWNERSHIP_STRENGTH (p. 1302)

6.127.2 Usage

Along with the **OWNERSHIP_STRENGTH** (p. 1302), this QoS policy specifies if **DDS_DataReader** (p. 615) entities can receive updates to the same instance (identified by its key) from multiple **DDS_DataWriter** (p. 485) entities at the same time.

There are two kinds of ownership, selected by the setting of the *kind*: **SHARED** and **EXCLUSIVE**.

6.127.2.1 SHARED ownership

DDS_SHARED_OWNERSHIP_QOS (p. 1302) 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 **DDS_Topic** (p. 175) will be able to access modifications from all **DDS_DataWriter** (p. 485) objects, subject to the settings of other QoS that may filter particular samples (e.g. the **TIME_BASED_FILTER** (p. 1329) or **HISTORY** (p. 1292) policy). In any case, there is no "filtering" of modifications made based on the identity of the **DDS_DataWriter** (p. 485) that causes the modification.

6.127.2.2 EXCLUSIVE ownership

DDS_EXCLUSIVE_OWNERSHIP_QOS (p. 1302) indicates that each instance of a data type can only be modified by one **DDS_DataWriter** (p. 485). In other words, at any point in time, a single **DDS_DataWriter** (p. 485) owns each instance and is the only one whose modifications will be visible to the **DDS_DataReader** (p. 615) objects. The owner is determined by selecting the **DDS_DataWriter** (p. 485) with the highest value of the **DDS_OwnershipStrengthQosPolicy::value** (p. 1869) that is currently alive, as defined by the **LIVELINESS** (p. 1296) policy, and has not violated its **DEADLINE** (p. 1272) contract with regards to the data instance.

Ownership can therefore change as a result of:

- a **DDS_DataWriter** (p. 485) in the system with a higher value of the strength that modifies the instance,
- a change in the strength value of the **DDS_DataWriter** (p. 485) that owns the instance, and
- a change in the liveliness of the **DDS_DataWriter** (p. 485) that owns the instance.
- a deadline with regards to the instance that is missed by the **DDS_DataWriter** (p. 485) that owns the instance.

The behavior of the system is as if the determination was made independently by each **DDS_DataReader** (p. 615). Each **DDS_DataReader** (p. 615) may detect the change of ownership at a different time. It is not a requirement that at a particular point in time all the **DDS_DataReader** (p. 615) objects for that **DDS_Topic** (p. 175) have a consistent picture of who owns each instance.

It is also not a requirement that the **DDS_DataWriter** (p. 485) objects are aware of whether they own a particular instance. There is no error or notification given to a **DDS_DataWriter** (p. 485) 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 **DDS_DataWriter** (p. 485) objects with the same strength modify the same instance. If this occurs RTI Connext will pick one of the **DDS_DataWriter** (p. 485) objects as the owner. It is not specified how the owner is selected. However, the algorithm used to select the owner guarantees that all **DDS_DataReader** (p. 615) objects will make the same choice of the particular **DDS_DataWriter** (p. 485) 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 **DDS_DataWriter** (p. 485) with higher same strength, or a new **DDS_DataWriter** (p. 485) 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 **DDS_DataWriter** (p. 485) as long as they affect instances whose values have not been set by the higher-strength **DDS_DataWriter** (p. 485).

6.127.3 Compatibility

The value of the **DDS_OwnershipQosPolicyKind** (p. 1301) offered must exactly match the one requested or else they are considered incompatible.

6.127.4 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. 493) and **FooDataWriter_dispose** (p. 502).

6.127.4.1 Ownership Resolution on Redundant Systems

It is expected that users may use DDS to set up redundant systems where multiple **DDS_DataWriter** (p. 485) entities are "capable" of writing the same instance. In this situation, the **DDS_DataWriter** (p. 485) 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. 1302) and arbitrate themselves by means of the **DDS_OwnershipStrengthQosPolicy** (p. 1868). Regardless of the scheme, the desired behavior from the **DDS_DataReader** (p. 615) point of view is that **DDS_DataReader** (p. 615) normally receives data from the primary unless the "primary" writer stops writing, in which case the **DDS_DataReader** (p. 615) starts to receive data from the secondary **DDS_DataWriter** (p. 485).

This approach requires some mechanism to detect that a **DDS_DataWriter** (p. 485) (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. 496).

Arbitrating from a **DDS_DataWriter** (p. 485) to one of a higher strength is simple and the decision can be taken autonomously by the **DDS_DataReader** (p. 615). Switching ownership from a higher strength **DDS_DataWriter** (p. 485) to one of a lower strength **DDS_DataWriter** (p. 485) requires that the **DDS_DataReader** (p. 615) can make a determination that the stronger **DDS_DataWriter** (p. 485) is "no longer writing the instance".

6.127.4.1.1 Case where the data is periodically updated This determination is reasonably simple when the data is being written periodically at some rate. The **DDS_DataWriter** (p.485) simply states its offered **DDS_DeadlineQosPolicy** (p.1684) (maximum interval between updates), and the **DDS_DataReader** (p.615) automatically monitors that the **DDS_DataWriter** (p.485) indeed updates the instance at least once per **DDS_DeadlineQosPolicy::period** (p.1685) set by the **DDS_DataReader** (p.615). If the deadline is missed, the **DDS_DataReader** (p.615) considers the **DDS_DataWriter** (p.485) "not alive" and automatically gives ownership to the next highest-strength **DDS_DataWriter** (p.485) that *is* alive.

6.127.4.1.2 Case where data is not periodically updated The case where the **DDS_DataWriter** (p.485) 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 **DDS_DataWriter** (p.485) is "alive" and for the **DDS_DataWriter** (p.485) to be alive it must fulfill its **DDS_LivelinessQosPolicy** (p.1823) 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.1297). The **DDS_DataWriter** (p.485) 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 **DDS_DataWriter** (p.485) process periodically writes data or calls **DDS_DomainParticipant_assert_liveliness** (p.136). However, this scenario requires that the **DDS_DataReader** (p.615) knows what instances are being "written" by the **DDS_DataWriter** (p.485). That is the only way that the **DDS_DataReader** (p.615) deduces the ownership of specific instances from the fact that the **DDS_DataWriter** (p.485) is still "alive". Hence the need for the **DDS_DataWriter** (p.485) to "register" and "unregister" instances. Note that while "registration" can be done lazily the first time the **DDS_DataWriter** (p.485) 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 **DDS_DataReader** (p.615).

6.127.4.2 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.1684) 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.1823) 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.715).

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).

6.127.4.3 Semantic Difference between unregister_instance and dispose

FooDataWriter_dispose (p. 502) is semantically different from **FooDataWriter_unregister_instance** (p. 493). **FooDataWriter_dispose** (p. 502) 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. 493) indicates that the writer is no longer taking responsibility for updating the value of the instance.

Deleting a **DDS_DataWriter** (p. 485) is equivalent to unregistering all the instances it was writing, but is *not* the same as "disposing" all the instances.

For a **DDS_Topic** (p. 175) with **DDS_EXCLUSIVE_OWNERSHIP_QOS** (p. 1302), 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 **DDS_DataWriter** (p. 485) 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 **DDS_Topic** (p. 175) with **DDS_EXCLUSIVE_OWNERSHIP_QOS** (p. 1302), 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.

6.127.5 Field Documentation

6.127.5.1 kind

DDS_OwnershipQosPolicyKind `DDS_OwnershipQosPolicy::kind`

<<**basic**>> (p. 783) <<**spec**>> (p. 790) The kind of ownership

[default] **DDS_SHARED_OWNERSHIP_QOS** (p. 1302)

Categories

Immutable (p. 914), **Matching** (p. 963), **Discovery Metadata** (p. 958), **Applied Per-Instance** (p. 956), **Data Availability** (p. 843)

6.128 DDS_OwnershipStrengthQosPolicy Struct Reference

Specifies the value of the strength used to arbitrate among multiple **DDS_DataWriter** (p. 485) objects that attempt to modify the same instance of a data type (identified by **DDS_Topic** (p. 175) + key).

Data Fields

- **DDS_Long** value

<<*basic*>> (p. 783) <<*spec*>> (p. 790) *The strength value used to arbitrate among multiple writers.*

6.128.1 Detailed Description

Specifies the value of the strength used to arbitrate among multiple **DDS_DataWriter** (p. 485) objects that attempt to modify the same instance of a data type (identified by **DDS_Topic** (p. 175) + key).

This policy only applies if the **OWNERSHIP** (p. 1301) policy is of kind **DDS_EXCLUSIVE_OWNERSHIP_QOS** (p. 1302).

Entity:

DDS_DataWriter (p. 485)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **YES** (p. 1245)

The value of the **OWNERSHIP_STRENGTH** (p. 1302) is used to determine the ownership of a data instance (identified by the key). The arbitration is performed by the **DDS_DataReader** (p. 615).

See also

EXCLUSIVE ownership (p. 1865)

6.128.2 Field Documentation

6.128.2.1 value

DDS_Long DDS_OwnershipStrengthQosPolicy::value

<<*basic*>> (p. 783) <<*spec*>> (p. 790) *The strength value used to arbitrate among multiple writers.*

[default] 0

[range] [0, 1 million]

Categories

Mutable (p. 949), **Discovery Metadata** (p. 958), **Applied Per-Instance** (p. 956), **Data Availability** (p. 843)

6.129 DDS_ParticipantBuiltinTopicData Struct Reference

Entry created when a DomainParticipant object is discovered.

Data Fields

- **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. 1012) Name value pair properties to be stored with DomainParticipant
- **DDS_ProtocolVersion_t rtps_protocol_version**
<<extension>> (p. 1012) Version number of the RTPS wire protocol used.
- struct **DDS_VendorId_t rtps_vendor_id**
<<extension>> (p. 1012) 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. 1012) Unicast locators used when individual entities do not specify unicast locators.
- struct **DDS_ProductVersion_t product_version**
<<extension>> (p. 1012) This is a vendor specific parameter. It gives the current version for rti-dds.
- struct **DDS_EntityNameQosPolicy participant_name**
<<extension>> (p. 1012) The participant name and role name.
- **DDS_DomainId_t domain_id**
<<extension>> (p. 1012) Domain ID associated with the discovered participant.
- struct **DDS_TransportInfoSeq transport_info**
*<<extension>> (p. 1012) A sequence of **DDS_TransportInfo_t** (p. 2052) containing information about each of the installed transports of the discovered participant.*
- struct **DDS_Duration_t reachability_lease_duration**
<<extension>> (p. 1012) Locator reachability lease duration.
- struct **DDS_PartitionQosPolicy partition**
<<extension>> (p. 1012) PartitionQosPolicy of the participant.
- **DDS_ParticipantTrustProtectionInfo trust_protection_info**
<<extension>> (p. 1012) Trust Plugins protection information associated with the discovered DomainParticipant.
- **DDS_ParticipantTrustAlgorithmInfo trust_algorithm_info**
<<extension>> (p. 1012) Trust Plugins algorithms associated with the discovered DomainParticipant.
- **DDS_Boolean partial_configuration**
*<<extension>> (p. 1012) Indicates whether a **DDS_ParticipantBuiltinTopicData** (p. 1870) only contains bootstrap-ping information.*

6.129.1 Detailed Description

Entry created when a DomainParticipant object is discovered.

Data associated with the built-in topic **DDS_PARTICIPANT_TOPIC_NAME** (p. 1092). It contains QoS policies and additional information that apply to the remote **DDS_DomainParticipant** (p. 77).

See also

DDS_PARTICIPANT_TOPIC_NAME (p. 1092)

DDS_ParticipantBuiltinTopicDataDataReader (p. 1092)

6.129.2 Field Documentation

6.129.2.1 key

```
DDS_BuiltinTopicKey_t DDS_ParticipantBuiltinTopicData::key
```

DCPS key to distinguish entries.

6.129.2.2 user_data

```
struct DDS_UserDataQosPolicy DDS_ParticipantBuiltinTopicData::user_data
```

Policy of the corresponding DomainParticipant.

6.129.2.3 property

```
struct DDS_PropertyQosPolicy DDS_ParticipantBuiltinTopicData::property
```

<<*extension*>> (p. 1012) Name value pair properties to be stored with DomainParticipant

6.129.2.4 rtps_protocol_version

```
DDS_ProtocolVersion_t DDS_ParticipantBuiltinTopicData::rtps_protocol_version
```

<<*extension*>> (p. 1012) Version number of the RTPS wire protocol used.

6.129.2.5 rtps_vendor_id

```
struct DDS_VendorId_t DDS_ParticipantBuiltinTopicData::rtps_vendor_id
```

<<**extension**>> (p. 1012) ID of vendor implementing the RTPS wire protocol.

6.129.2.6 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

6.129.2.7 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. 1872) 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

6.129.2.8 default_unicast_locators

`struct DDS_LocatorSeq DDS_ParticipantBuiltinTopicData::default_unicast_locators`

<<*extension*>> (p. 1012) Unicast locators used when individual entities do not specify unicast locators.

6.129.2.9 product_version

`struct DDS_ProductVersion_t DDS_ParticipantBuiltinTopicData::product_version`

<<*extension*>> (p. 1012) This is a vendor specific parameter. It gives the current version for rti-dds.

6.129.2.10 participant_name

`struct DDS_EntityNameQosPolicy DDS_ParticipantBuiltinTopicData::participant_name`

<<*extension*>> (p. 1012) The participant name and role name.

This parameter contains the name and the role name of the discovered participant.

6.129.2.11 domain_id

`DDS_DomainId_t DDS_ParticipantBuiltinTopicData::domain_id`

<<*extension*>> (p. 1012) Domain ID associated with the discovered participant.

6.129.2.12 transport_info

```
struct DDS_TransportInfoSeq DDS_ParticipantBuiltinTopicData::transport_info
```

<<**extension**>> (p. 1012) A sequence of **DDS_TransportInfo_t** (p. 2052) containing information about each of the installed transports of the discovered participant.

This parameter contains a sequence of **DDS_TransportInfo_t** (p. 2052) containing the `class_id` and `message_size_↵` `max` for all installed transports of the discovered participant. The maximum number of **DDS_TransportInfo_t** (p. 2052) 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. 1755).

6.129.2.13 reachability_lease_duration

```
struct DDS_Duration_t DDS_ParticipantBuiltinTopicData::reachability_lease_duration
```

<<**extension**>> (p. 1012) 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.

6.129.2.14 partition

```
struct DDS_PartitionQosPolicy DDS_ParticipantBuiltinTopicData::partition
```

<<**extension**>> (p. 1012) PartitionQosPolicy of the participant.

6.129.2.15 trust_protection_info

```
DDS_ParticipantTrustProtectionInfo DDS_ParticipantBuiltinTopicData::trust_protection_info
```

<<**extension**>> (p. 1012) 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`.

6.129.2.16 trust_algorithm_info

DDS_ParticipantTrustAlgorithmInfo DDS_ParticipantBuiltinTopicData::trust_algorithm_info

<<**extension**>> (p. 1012) 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](#).

6.129.2.17 partial_configuration

DDS_Boolean DDS_ParticipantBuiltinTopicData::partial_configuration

<<**extension**>> (p. 1012) Indicates whether a **DDS_ParticipantBuiltinTopicData** (p. 1870) only contains bootstrapping information.

If this is **DDS_BOOLEAN_TRUE** (p. 1201), the **DDS_ParticipantBuiltinTopicData** (p. 1870) only contains bootstrapping information. If it is **DDS_BOOLEAN_FALSE** (p. 1201), it contains both bootstrapping and configuration information. The following fields are valid when this is set to **DDS_BOOLEAN_TRUE** (p. 1201):

- **DDS_ParticipantBuiltinTopicData::key** (p. 1871)
- **DDS_ParticipantBuiltinTopicData::property** (p. 1871) (only dds.domain_participant.domain_tag is valid if set)
- **DDS_ParticipantBuiltinTopicData::rtps_protocol_version** (p. 1871)
- **DDS_ParticipantBuiltinTopicData::rtps_vendor_id** (p. 1871)
- **DDS_ParticipantBuiltinTopicData::product_version** (p. 1873)
- **DDS_ParticipantBuiltinTopicData::domain_id** (p. 1873)
- **DDS_ParticipantBuiltinTopicData::transport_info** (p. 1873)
- **DDS_ParticipantBuiltinTopicData::partition** (p. 1874)
- **DDS_ParticipantBuiltinTopicData::trust_protection_info** (p. 1874)
- **DDS_ParticipantBuiltinTopicData::trust_algorithm_info** (p. 1874)

All other fields are invalid.

This field will only be set to **DDS_BOOLEAN_TRUE** (p. 1201) if a participant using **DDS_DISCOVERYCONFIG_BUILTIN_SPDP2** (p. 1281) receives a bootstrap message and **DDS_DiscoveryConfigQosPolicy::ignore_default_domain_announcements** (p. 1705) is set to **DDS_BOOLEAN_FALSE** (p. 1201) (non-default).

If a participant is using **DDS_DISCOVERYCONFIG_BUILTIN_SPDP** (p. 1280), this field will always be set to **DDS_BOOLEAN_FALSE** (p. 1201).

6.130 DDS_ParticipantBuiltinTopicDataSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_ParticipantBuiltinTopicData** (p. 1870) > .

6.130.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_ParticipantBuiltinTopicData** (p. 1870) > .

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_ParticipantBuiltinTopicData (p. 1870)

6.131 DDS_ParticipantBuiltinTopicDataTypeSupport Struct Reference

Instantiates **TypeSupport** < **DDS_ParticipantBuiltinTopicData** (p. 1870) > .

6.131.1 Detailed Description

Instantiates **TypeSupport** < **DDS_ParticipantBuiltinTopicData** (p. 1870) > .

Instantiates:

<<*generic*>> (p. 1013) **FooTypeSupport** (p. 2114)

See also

DDS_ParticipantBuiltinTopicData (p. 1870)

6.132 DDS_ParticipantTrustAlgorithmInfo Struct Reference

Trust Plugins algorithm information associated with the discovered DomainParticipant.

Data Fields

- **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.

6.132.1 Detailed Description

Trust Plugins algorithm information associated with the discovered DomainParticipant.

6.132.2 Field Documentation

6.132.2.1 signature

DDS_ParticipantTrustSignatureAlgorithmInfo DDS_ParticipantTrustAlgorithmInfo::signature

Information regarding algorithms for validation of data and metadata exchanged by the DomainParticipant.

6.132.2.2 key_establishment

DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo DDS_ParticipantTrustAlgorithmInfo::key_establishment

Information regarding algorithms for transformation of data and metadata exchanged between two DomainParticipants.

6.132.2.3 interceptor

DDS_ParticipantTrustInterceptorAlgorithmInfo DDS_ParticipantTrustAlgorithmInfo::interceptor

Information regarding algorithms for interception of data and metadata exchanged by the DomainParticipant.

6.133 DDS_ParticipantTrustInterceptorAlgorithmInfo Struct Reference

Trust Plugins interception algorithm information associated with the discovered DomainParticipant.

Data Fields

- **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. 1786) is not serialized.*

6.133.1 Detailed Description

Trust Plugins interception algorithm information associated with the discovered DomainParticipant.

6.133.2 Field Documentation

6.133.2.1 supported_mask

DDS_TrustAlgorithmSet DDS_ParticipantTrustInterceptorAlgorithmInfo::supported_mask

Trust Plugins algorithms supported for interception of data and metadata exchanged by the DomainParticipant.

6.133.2.2 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.

6.133.2.3 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.

6.133.2.4 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. 1786) is not serialized.

6.134 DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo Struct Reference

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

Data Fields

- **DDS_TrustAlgorithmRequirements shared_secret**

Trust Plugins key establishment algorithm requirements of the DomainParticipant in the context of deriving a shared secret.

6.134.1 Detailed Description

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

6.134.2 Field Documentation

6.134.2.1 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.

6.135 DDS_ParticipantTrustProtectionInfo Struct Reference

Trust Plugins Protection information associated with the discovered DomainParticipant.

Data Fields

- **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.

6.135.1 Detailed Description

Trust Plugins Protection information associated with the discovered DomainParticipant.

6.135.2 Field Documentation

6.135.2.1 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](#).

6.135.2.2 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](#).

6.136 DDS_ParticipantTrustSignatureAlgorithmInfo Struct Reference

Trust Plugins signature algorithm information associated with the discovered DomainParticipant.

Data Fields

- **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.

6.136.1 Detailed Description

Trust Plugins signature algorithm information associated with the discovered DomainParticipant.

6.136.2 Field Documentation

6.136.2.1 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.

6.136.2.2 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.

6.137 DDS_PartitionQosPolicy Struct Reference

<<spec>> (p. 790) Set of strings that introduces logical partitions in **DDS_DomainParticipant** (p. 77), **DDS_Publisher** (p. 443), or **DDS_Subscriber** (p. 572) entities.

Data Fields

- struct **DDS_StringSeq** name
<<basic>> (p. 783) **<<spec>>** (p. 790) *A list of partition names*

6.137.1 Detailed Description

<<spec>> (p. 790) Set of strings that introduces logical partitions in **DDS_DomainParticipant** (p. 77), **DDS_Publisher** (p. 443), or **DDS_Subscriber** (p. 572) 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 **DDS_DataWriter** (p. 485) within a **DDS_Publisher** (p. 443) only communicates with a **DDS_DataReader** (p. 615) in a **DDS_Subscriber** (p. 572) if (in addition to matching the **DDS_Topic** (p. 175) and having compatible QoS) the **DDS_Publisher** (p. 443) and **DDS_Subscriber** (p. 572) have a common partition name string.
- **DDS_DomainParticipant** (p. 77) 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:

DDS_Publisher (p. 443), **DDS_Subscriber** (p. 572), **DDS_DomainParticipant** (p. 77)

Properties:

RxO (p. 1245) = NO

Changeable (p. 1245) = YES (p. 1245)

6.137.2 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 **DDS_DomainParticipant** (p. 77), **DDS_Publisher** (p. 443), or **DDS_Subscriber** (p. 572) 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 Connex 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 **DDS_Entity** (p. 1360) 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 Connex itself to see entities in a domain it does not belong to.
- Second, a **DDS_Entity** (p. 1360) can only belong to one domain whereas a **DDS_Entity** (p. 1360) can be in multiple partitions.
- Finally, as far as RTI Connex is concerned, each unique data instance is identified by the tuple (**DomainID**, **domain tag**, **DDS_Topic** (p. 175), **key**). Therefore two **DDS_Entity** (p. 1360) 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 QoS Policy" section of the `Core Libraries User's Manual`.

6.137.3 Field Documentation

6.137.3.1 name

```
struct DDS_StringSeq DDS_PartitionQosPolicy::name
```

<<**basic**>> (p. 783) <<**spec**>> (p. 790) 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. 1222) when setting a **DDS_Publisher** (p. 443)'s or **DDS_Subscriber** (p. 572)'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. 1881) is specified on a domain basis in **DDS_DomainParticipantResourceLimitsQosPolicy::max_partitions** (p. 1748). 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. 1881) is specified on a domain basis in **DDS_DomainParticipantResourceLimitsQosPolicy::max_partition_cumulative_characters** (p. 1748).

The memory for the strings in this sequence is managed according to the conventions described in **String Conventions** (p. 1527). In particular, be careful to avoid a situation in which RTI Connexx allocates a string on your behalf and you then reuse that string in such a way that RTI Connexx 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 Connexx will assume the entity to be in default partition (empty string partition "").

[range] List of partition name with above restrictions

Categories

Mutable (p. 949), **Discovery Isolation** (p. 858), **Matching** (p. 963), **Discovery Metadata** (p. 958)

6.138 DDS_PersistentStorageSettings Struct Reference

Configures durable writer history and durable reader state.

Data Fields

- **DDS_Boolean enable**
 <<extension>> (p. 795) Enables durable writer history in a **DDS_DataWriter** (p. 485) and durable reader state in a **DDS_DataReader** (p. 615).
- char * **file_name**
 <<extension>> (p. 795) The file name where the durable writer history or durable reader state will be stored.
- char * **trace_file_name**
 <<extension>> (p. 795) 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. 795) Sets the journal mode of the persistent storage.
- **DDS_PersistentSynchronizationKind synchronization_kind**
 <<extension>> (p. 795) Sets the level of synchronization with the physical disk.
- **DDS_Boolean vacuum**
 <<extension>> (p. 795) Sets the auto-vacuum status of the storage.
- **DDS_Boolean restore**
 <<extension>> (p. 795) Indicates if the persisted writer history or reader state must be restored.
- struct **DDS_AllocationSettings_t writer_instance_cache_allocation**
 <<extension>> (p. 795) Configures the resource limits associated with the instance durable writer history cache.
- struct **DDS_AllocationSettings_t writer_sample_cache_allocation**
 <<extension>> (p. 795) Configures the resource limits associated with the sample durable writer history cache.
- **DDS_Boolean writer_memory_state**
 <<extension>> (p. 795) 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. 795) Controls how often the reader state is stored into the database.

6.138.1 Detailed Description

Configures durable writer history and durable reader state.

In a **DDS_DataWriter** (p. 485), 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 **DDS_DataReader** (p. 615), 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. 1759)

6.138.2 Field Documentation

6.138.2.1 enable

`DDS_Boolean DDS_PersistentStorageSettings::enable`

<<**extension**>> (p. 795) Enables durable writer history in a **DDS_DataWriter** (p. 485) and durable reader state in a **DDS_DataReader** (p. 615).

When this field is set to **DDS_BOOLEAN_TRUE** (p. 1201), 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. 1201)

Categories

Immutable (p. 914), **Durability** (p. 860)

6.138.2.2 file_name

`char* DDS_PersistentStorageSettings::file_name`

<<**extension**>> (p. 795) 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. 1887) is set to **DDS_BOOLEAN_TRUE** (p. 1201), 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. 1597) and **DDS_DataWriterProtocolQosPolicy::virtual_guid** (p. 1650) 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. 914), **Durability** (p. 860)

6.138.2.3 trace_file_name

```
char* DDS_PersistentStorageSettings::trace_file_name
```

<<**extension**>> (p. 795) 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. 914), **Durability** (p. 860)

6.138.2.4 journal_kind

```
DDS_PersistentJournalKind DDS_PersistentStorageSettings::journal_kind
```

<<**extension**>> (p. 795) Sets the journal mode of the persistent storage.

[default] DDS_WAL_PERSISTENT_JOURNAL (p. 1286)

Categories

Immutable (p. 914), **Durability** (p. 860)

6.138.2.5 synchronization_kind

```
DDS_PersistentSynchronizationKind DDS_PersistentStorageSettings::synchronization_kind
```

<<**extension**>> (p. 795) Sets the level of synchronization with the physical disk.

[default] DDS_NORMAL_PERSISTENT_SYNCHRONIZATION (p. 1287)

Categories

Immutable (p. 914), **Durability** (p. 860)

6.138.2.6 vacuum

DDS_Boolean DDS_PersistentStorageSettings::vacuum

<<**extension**>> (p. 795) Sets the auto-vacuum status of the storage.

When auto-vacuum is **DDS_BOOLEAN_TRUE** (p. 1201), the storage files will be compacted automatically with every transaction.

When auto-vacuum is **DDS_BOOLEAN_FALSE** (p. 1201), after data is deleted from the storage files, the files remain the same size.

[default] DDS_BOOLEAN_TRUE (p. 1201)

Categories

Immutable (p. 914), **Durability** (p. 860)

6.138.2.7 restore

DDS_Boolean DDS_PersistentStorageSettings::restore

<<**extension**>> (p. 795) Indicates if the persisted writer history or reader state must be restored.

For a **DDS_DataWriter** (p. 485), this field indicates whether or not the persisted writer history must be restored once the DataWriter is restarted.

For a **DDS_DataReader** (p. 615), this field indicates whether or not the persisted reader state must be restored once the DataReader is restarted.

[default] DDS_BOOLEAN_TRUE (p. 1201)

Categories

Immutable (p. 914), **Durability** (p. 860)

6.138.2.8 writer_instance_cache_allocation

```
struct DDS_AllocationSettings_t DDS_PersistentStorageSettings::writer_instance_cache_allocation
```

<<**extension**>> (p. 795) Configures the resource limits associated with the instance durable writer history cache.

This field only applies to **DDS_DataWriter** (p. 485) entities.

To minimize the number of accesses to the persisted storage, RTI Connexx 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. 1951) and **DDS_ResourceLimitsQosPolicy::initial_instances** (p. 1952).

If **DDS_PersistentStorageSettings::writer_memory_state** (p. 1889) is set to **DDS_BOOLEAN_TRUE** (p. 1201), then the value of **DDS_AllocationSettings_t::max_count** (p. 1538) is set to **DDS_LENGTH_UNLIMITED** (p. 1326), overwriting any value set by the user.

DDS_AllocationSettings_t::incremental_count (p. 1538) is ignored.

[range] **DDS_AllocationSettings_t::max_count** (p. 1538) in interval [1, INT_MAX], **DDS_LENGTH_AUTO** (p. 1321), or **DDS_LENGTH_UNLIMITED** (p. 1326). **DDS_AllocationSettings_t::initial_count** (p. 1538) in interval [1, INT_MAX], or **DDS_LENGTH_AUTO** (p. 1321).

DDS_LENGTH_AUTO (p. 1321) means that the value will be set to the equivalent value of **DDS_ResourceLimitsQosPolicy** (p. 1949).

[default] **DDS_AllocationSettings_t::max_count** (p. 1538) = **DDS_LENGTH_AUTO** (p. 1321) (= **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1951)) and **DDS_AllocationSettings_t::initial_count** (p. 1538) = **DDS_LENGTH_AUTO** (p. 1321) (= **DDS_ResourceLimitsQosPolicy::initial_instances** (p. 1952)).

Categories

Immutable (p. 914), **Durability** (p. 860), **Sample Memory Management** (p. 902)

6.138.2.9 writer_sample_cache_allocation

```
struct DDS_AllocationSettings_t DDS_PersistentStorageSettings::writer_sample_cache_allocation
```

<<**extension**>> (p. 795) Configures the resource limits associated with the sample durable writer history cache.

This field only applies to **DDS_DataWriter** (p. 485) entities.

To minimize the number of accesses to the persisted storage, RTI Connexx 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. 1951) and **DDS_ResourceLimitsQosPolicy::initial_samples** (p. 1952).

DDS_AllocationSettings_t::incremental_count (p. 1538) is ignored.

[range] **DDS_AllocationSettings_t::max_count** (p. 1538) in interval [1, INT_MAX], **DDS_LENGTH_AUTO** (p. 1321), or **DDS_LENGTH_UNLIMITED** (p. 1326). **DDS_AllocationSettings_t::initial_count** (p. 1538) in interval [1, INT_MAX], or **DDS_LENGTH_AUTO** (p. 1321).

DDS_LENGTH_AUTO (p. 1321) means that the value will be set to the equivalent value of **DDS_ResourceLimitsQosPolicy** (p. 1949).

[default] **DDS_AllocationSettings_t::max_count** (p. 1538) = 32 and **DDS_AllocationSettings_t::initial_count** (p. 1538) = 32.

Categories

Immutable (p. 914), **Durability** (p. 860), **Sample Memory Management** (p. 902)

6.138.2.10 writer_memory_state

DDS_Boolean DDS_PersistentStorageSettings::writer_memory_state

<<**extension**>> (p. 795) 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 **DDS_DataWriter** (p. 485) entities.

If this field is set to **DDS_BOOLEAN_TRUE** (p. 1201), then **DDS_AllocationSettings_t::max_count** (p. 1538) in **DDS_PersistentStorageSettings::writer_instance_cache_allocation** (p. 1887) is set to **DDS_LENGTH_UNLIMITED** (p. 1326), 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. 1201), 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. 1201) when the DataWriter is configured with **DDS_↵ReliabilityQosPolicy::acknowledgment_kind** (p. 1939) set to **DDS_APPLICATION_AUTO_ACKNOWLEDGMENT_↵_MODE** (p. 1323) or **DDS_APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE** (p. 1324), or **DDS_Availability_↵QosPolicy::enable_required_subscriptions** (p. 1550) is set to **DDS_BOOLEAN_TRUE** (p. 1201).

[default] **DDS_BOOLEAN_TRUE** (p. 1201)

Categories

Immutable (p. 914), **Durability** (p. 860), **Middleware Memory Management** (p. 878)

6.138.2.11 reader_checkpoint_frequency

DDS_UnsignedLong DDS_PersistentStorageSettings::reader_checkpoint_frequency

<<**extension**>> (p. 795) Controls how often the reader state is stored into the database.

This field only applies to **DDS_DataReader** (p. 615) 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. 914), **Durability** (p. 860)

6.139 DDS_PresentationQosPolicy Struct Reference

<<*spec*>> (p. 790) Specifies how the samples representing changes to data instances are presented to a subscribing application.

Data Fields

- **DDS_PresentationQosPolicyAccessScopeKind access_scope**
 <<*spec*>> (p. 790) Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.
- **DDS_Boolean coherent_access**
 <<*spec*>> (p. 790) Specifies support for coherent access. Controls whether coherent access is supported within the scope *access_scope*.
- **DDS_Boolean ordered_access**
 <<*spec*>> (p. 790) 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. 795) Indicates whether or not a **DDS_DataReader** (p. 615) 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.

6.139.1 Detailed Description

<<*spec*>> (p. 790) 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:

DDS_Publisher (p. 443), **DDS_Subscriber** (p. 572)

Status:

DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 1229), **DDS_REQUESTED_INCOMPATIBLE_QOS_↵ STATUS** (p. 1229)

Properties:

RxO (p. 1245) = YES
Changeable (p. 1245) = **UNTIL ENABLE** (p. 1245)

6.139.2 Usage

A **DDS_DataReader** (p. 615) will usually receive data in the order that it was sent by a **DDS_DataWriter** (p. 485), and the data is presented to the **DDS_DataReader** (p. 615) 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 Connexx will preserve the groupings of changes made by a publishing application by means of the operations **DDS_Publisher_begin_coherent_changes** (p. 459) and **DDS_Publisher_end_coherent_changes** (p. 460).
- `ordered_access` controls whether RTI Connexx 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 **DDS_Subscriber** (p. 572) controls the maximum extent of coherent changes. The behavior is as follows:

- If `access_scope` is set to **DDS_INSTANCE_PRESENTATION_QOS** (p. 1305) (the default), the behavior is the same as **DDS_TOPIC_PRESENTATION_QOS** (p. 1305).
- If `access_scope` is set to **DDS_TOPIC_PRESENTATION_QOS** (p. 1305), then coherent changes (indicated by their enclosure within calls to **DDS_Publisher_begin_coherent_changes** (p. 459) and **DDS_Publisher_end_coherent_changes** (p. 460)) will be made available as a unit to each remote **DDS_DataReader** (p. 615) independently. That is, changes made to instances within each individual **DDS_DataWriter** (p. 485) will be presented as a unit. They will not be grouped with changes made to instances belonging to a different **DDS_DataWriter** (p. 485).
- If `access_scope` is set to **DDS_GROUP_PRESENTATION_QOS** (p. 1305), then coherent changes made to instances through a set of **DDS_DataWriter** (p. 485) entities attached to a common **DDS_Publisher** (p. 443) will be made available as a unit to the **DDS_DataReader** (p. 615) entities within a **DDS_Subscriber** (p. 572).

If `ordered_access` is set, then the `access_scope` set on the **DDS_Subscriber** (p. 572) controls the maximum extent to which order will be preserved by RTI Connexx.

- If `access_scope` is set to **DDS_INSTANCE_PRESENTATION_QOS** (p. 1305) (the lowest level), then the relative order of DDS samples sent by a **DDS_DataWriter** (p. 485) 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 **DDS_DataReader** (p. 615) 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 **DDS_DataWriter** (p. 485).

- If `access_scope` is set to **DDS_TOPIC_PRESENTATION_QOS** (p. 1305), changes (creations, deletions, modifications) made by a single **DDS_DataWriter** (p. 485) to one or more instances are presented to a **DDS_DataReader** (p. 615) in the same order in which they occur. Changes made to instances through different **DDS_DataWriter** (p. 485) 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 **DDS_DataWriter** (p. 485) objects attached to the same **DDS_Publisher** (p. 443).
- Finally, if `access_scope` is set to **DDS_GROUP_PRESENTATION_QOS** (p. 1305), changes made to instances via **DDS_DataWriter** (p. 485) entities attached to the same **DDS_Publisher** (p. 443) object are presented to the **DDS_DataReader** (p. 615) entities within a **DDS_Subscriber** (p. 572) in the same order in which they occur. For this to happen, the application must take the samples using the Subscriber's **DDS_Subscriber_begin_access** (p. 587) and **DDS_Subscriber_end_access** (p. 588) operations; see the "Ordered Access" section in the "PRESENTATION QosPolicy" section of the [Core Libraries User's Manual](http://manual.rti-connextdds-corelibraries-usersmanual.pdf).

If `access_scope` is set to **DDS_HIGHEST_OFFERED_PRESENTATION_QOS** (p. 1305) on the **DDS_Subscriber** (p. 572), then the **DDS_Subscriber** (p. 572) will use the `access_scope` offered by the remote **DDS_Publisher** (p. 443).

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 **DDS_Subscriber** (p. 572) **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 **DDS_DataReader** (p. 615) objects.

For **DDS_GROUP_PRESENTATION_QOS** (p. 1305) the subscribing application must use the APIs **DDS_Subscriber_begin_access** (p. 587), **DDS_Subscriber_end_access** (p. 588) and **DDS_Subscriber_get_datareaders** (p. 588) 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 **DDS_Publisher** (p. 443).

The field **DDS_SampleInfo::coherent_set_info** (p. 1995) 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. 1568) 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 **DDS_DataWriter** (p. 485) 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. 1894) to **DDS_BOOLEAN_FALSE** (p. 1201), 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. 1568) set to **DDS_BOOLEAN_TRUE** (p. 1201).

6.139.3 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. 1305). For the purposes of this inequality, the values of `access_scope` are considered ordered such that **DDS_INSTANCE_PRESENTATION_QOS** (p. 1305) < **DDS_TOPIC_PRESENTATION_QOS** (p. 1305) < **DDS_GROUP_PRESENTATION_QOS** (p. 1305).

- requested `coherent_access` is **DDS_BOOLEAN_FALSE** (p.1201), or else both offered and requested `coherent_access` are **DDS_BOOLEAN_TRUE** (p.1201).
- requested `ordered_access` is **DDS_BOOLEAN_FALSE** (p.1201), or else both offered and requested `ordered_access` are **DDS_BOOLEAN_TRUE** (p.1201).

6.139.4 Field Documentation

6.139.4.1 access_scope

DDS_PresentationQosPolicyAccessScopeKind `DDS_PresentationQosPolicy::access_scope`

<<*spec*>> (p. 790) Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.

[default] **DDS_INSTANCE_PRESENTATION_QOS** (p. 1305)

Categories

Immutable (p.914), **Matching** (p.963), **Discovery Metadata** (p.958), **Applied Per-Instance** (p.956), **Sample Ordering** (p.905), **Data Availability** (p.843)

6.139.4.2 coherent_access

DDS_Boolean `DDS_PresentationQosPolicy::coherent_access`

<<*spec*>> (p. 790) 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. 1323)).

[default] **DDS_BOOLEAN_FALSE** (p. 1201)

Categories

Immutable (p.914), **Matching** (p.963), **Discovery Metadata** (p.958), **Sample Ordering** (p.905), **Data Availabil-**
ity (p.843)

6.139.4.3 ordered_access

DDS_Boolean DDS_PresentationQosPolicy::ordered_access

<<**spec**>> (p. 790) 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. 1201)

Categories

Immutable (p. 914), **Matching** (p. 963), **Discovery Metadata** (p. 958), **Applied Per-Instance** (p. 956), **Sample Ordering** (p. 905), **Data Availability** (p. 843)

6.139.4.4 drop_incomplete_coherent_set

DDS_Boolean DDS_PresentationQosPolicy::drop_incomplete_coherent_set

<<**extension**>> (p. 795) Indicates whether or not a **DDS_DataReader** (p. 615) 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. 1201), you can change this behavior.

Samples from an incomplete coherent set have **DDS_CoherentSetInfo_t::incomplete_coherent_set** (p. 1568) set to **DDS_BOOLEAN_TRUE** (p. 1201) in **DDS_SampleInfo::coherent_set_info** (p. 1995).

[default] **DDS_BOOLEAN_TRUE** (p. 1201)

Categories

Immutable (p. 914), **Sample Ordering** (p. 905), **Data Availability** (p. 843)

6.140 DDS_PrintFormatProperty Struct Reference

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

Data Fields

- **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.

6.140.1 Detailed Description

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

To ensure that new objects are initialized to known values, assign them with the static initializer **DDS_PrintFormatProperty_INITIALIZER** (p. 172).

See also

DDS_PrintFormatProperty_INITIALIZER (p. 172)

Examples

HelloWorldPlugin.c.

6.140.2 Field Documentation

6.140.2.1 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. 186)

[default] DDS_DEFAULT_PRINT_FORMAT (p. 186)

6.140.2.2 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. 1201) 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. 1201) 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. 1201)

6.140.2.3 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. 1201) 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. 1201) 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. 1201)

6.140.2.4 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. 1895) is **DDS_XML_PRINT_FORMAT** (p. 186) or **DDS_JSON_PRINT_FORMAT** (p. 186)

If the value is **DDS_BOOLEAN_TRUE** (p. 1201) and the kind is **DDS_XML_PRINT_FORMAT** (p. 186) 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. 1201) and the kind is **DDS_JSON_PRINT_FORMAT** (p. 186) 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. 1201) the elements described above will not be included in the output. For example:

```
<myLong>5</myLong>
"myLong":5
```

[default] DDS_BOOLEAN_TRUE (p. 1201)

6.141 DDS_ProductVersion_t Struct Reference

<<*extension*>> (p. 1012) Type used to represent the current version of RTI Connex.

Data Fields

- **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.

6.141.1 Detailed Description

<<*extension*>> (p. 1012) Type used to represent the current version of RTI Connex.

6.141.2 Field Documentation

6.141.2.1 major

DDS_Char DDS_ProductVersion_t::major

Major product version.

6.141.2.2 minor

DDS_Char DDS_ProductVersion_t::minor

Minor product version.

6.141.2.3 release

DDS_Char DDS_ProductVersion_t::release

Release letter for product version.

6.141.2.4 revision

DDS_Char DDS_ProductVersion_t::revision

Revision number of product.

6.142 DDS_ProfileQosPolicy Struct Reference

<<**extension**>> (p. 795) Configures the way that XML documents containing QoS profiles are loaded by RTI Connex.

Data Fields

- struct **DDS_StringSeq string_profile**
 <<**basic**>> (p. 783) <<**extension**>> (p. 795) Sequence of strings containing a XML document to load.
- struct **DDS_StringSeq url_profile**
 <<**basic**>> (p. 783) <<**extension**>> (p. 795) Sequence of **URL groups** (p. 973) containing a set of XML documents to load.
- **DDS_Boolean ignore_user_profile**
 <<**basic**>> (p. 783) <<**extension**>> (p. 795) Ignores the file **USER_QOS_PROFILES.xml** in the current working directory.
- **DDS_Boolean ignore_environment_profile**
 <<**basic**>> (p. 783) <<**extension**>> (p. 795) Ignores the value of the **NDDS_QOS_PROFILES** (p. 973).
- **DDS_Boolean ignore_resource_profile**
 <<**basic**>> (p. 783) <<**extension**>> (p. 795) Ignores the file **NDDS_QOS_PROFILES.xml**.

6.142.1 Detailed Description

<<**extension**>> (p. 795) Configures the way that XML documents containing QoS profiles are loaded by RTI Connex.

All QoS values for Entities can be configured in QoS profiles defined in XML documents. XML documents can be passed to RTI Connex 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 Connex, and the locations specified by an environment variable.

You may disable any or all of these default locations using the Profile QoS policy.

Entity:

DDS_DomainParticipantFactory (p. 32)

Properties:

RxO (p. 1245) = NO

Changeable (p. 1245) = **Changeable** (p. 1245)

6.142.2 Field Documentation

6.142.2.1 string_profile

```
struct DDS_StringSeq DDS_ProfileQosPolicy::string_profile
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) 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. 914), **QoS Profile Management** (p. 893)

6.142.2.2 url_profile

```
struct DDS_StringSeq DDS_ProfileQosPolicy::url_profile
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Sequence of **URL groups** (p. 973) 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. 914), **QoS Profile Management** (p. 893)

6.142.2.3 ignore_user_profile

```
DDS_Boolean DDS_ProfileQosPolicy::ignore_user_profile
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Ignores the file USER_QOS_PROFILES.xml in the current working directory.

When this field is set to **DDS_BOOLEAN_TRUE** (p. 1201), the QoS profiles contained in the file USER_QOS_↵PROFILES.xml in the current working directory will be ignored.

[default] **DDS_BOOLEAN_FALSE** (p. 1201)

Categories

Immutable (p. 914), **QoS Profile Management** (p. 893)

6.142.2.4 ignore_environment_profile

DDS_Boolean DDS_ProfileQosPolicy::ignore_environment_profile

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Ignores the value of the **NDDS_QOS_PROFILES** (p. 973).

When this field is set to **DDS_BOOLEAN_TRUE** (p. 1201), the value of the environment variable **NDDS_QOS_PROFILES** will be ignored.

[default] **DDS_BOOLEAN_FALSE** (p. 1201)

Categories

Immutable (p. 914), **QoS Profile Management** (p. 893)

6.142.2.5 ignore_resource_profile

DDS_Boolean DDS_ProfileQosPolicy::ignore_resource_profile

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Ignores the file **NDDS_QOS_PROFILES.xml**.

When this field is set to **DDS_BOOLEAN_TRUE** (p. 1201), the QoS profiles contained in the file **NDDS_QOS_PROFILES.xml** in **\$NDDSHOME/resource/xml** will be ignored.

[default] **DDS_BOOLEAN_FALSE** (p. 1201)

Categories

Immutable (p. 914), **QoS Profile Management** (p. 893)

6.143 DDS_Property_t Struct Reference

Properties are name/value pairs objects.

Data Fields

- char * **name**
Property name.
- char * **value**
Property value.
- **DDS_Boolean propagate**
Indicates if the property must be propagated on discovery.

6.143.1 Detailed Description

Properties are name/value pairs objects.

6.143.2 Field Documentation

6.143.2.1 name

```
char* DDS_Property_t::name
```

Property name.

It must be a NULL-terminated string allocated with **DDS_String_alloc** (p. 1528) or **DDS_String_dup** (p. 1528).

6.143.2.2 value

```
char* DDS_Property_t::value
```

Property value.

It must be a NULL-terminated string allocated with **DDS_String_alloc** (p. 1528) or **DDS_String_dup** (p. 1528).

6.143.2.3 propagate

```
DDS_Boolean DDS_Property_t::propagate
```

Indicates if the property must be propagated on discovery.

6.144 DDS_PropertyQosPolicy Struct Reference

<<**extension**>> (p. 795) Stores name/value(string) pairs that can be used to configure certain parameters of RTI Connex 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.

Data Fields

- struct **DDS_PropertySeq** value

<<**basic**>> (p. 783) <<**extension**>> (p. 795) *Sequence of properties.*

6.144.1 Detailed Description

<<**extension**>> (p. 795) Stores name/value(string) pairs that can be used to configure certain parameters of RTI Connex 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:

DDS_DomainParticipant (p. 77) **DDS_DataReader** (p. 615) **DDS_DataWriter** (p. 485)

Properties:

RxO (p. 1245) = N/A;
Changeable (p. 1245) = **YES** (p. 1245)

See also

DDS_DomainParticipant_get_builtin_subscriber (p. 129)

6.144.2 Usage

The **PROPERTY** QoS policy can be used to associate a set of properties in the form of (name, value) pairs with a **DDS_DataReader** (p. 615), **DDS_DataWriter** (p. 485), or **DDS_DomainParticipant** (p. 77). This is similar to the **DDS_UserDataQosPolicy** (p. 2086), 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. 779)
- Durable Reader State, see **Configuring Durable Reader State** (p. 781)
- Builtin Transport Plugins, see **UDIPv4 Transport Property Names in Property QoS Policy of Domain Participant** (p. 1044), **UDIPv6 Transport Property Names in Property QoS Policy of Domain Participant** (p. 1060), and **Shared Memory Transport Property Names in Property QoS Policy of Domain Participant** (p. 1038)
- Extension Transport Plugins, see **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 727)
- **Clock Selection** (p. 25)

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 Connex 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. 2086) or **DDS_GroupDataQosPolicy** (p. 1803)).

6.144.2.1 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. 1306) page.

6.144.3 Field Documentation

6.144.3.1 value

```
struct DDS_PropertySeq DDS_PropertyQoSPolicy::value
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Sequence of properties.

[default] An empty list.

Categories

Mutable (p. 949), **Discovery Metadata** (p. 958), **Application-Defined Metadata** (p. 836)

6.145 DDS_PropertySeq Struct Reference

Declares IDL sequence < **DDS_Property_t** (p. 1900) >

6.145.1 Detailed Description

Declares IDL sequence < **DDS_Property_t** (p. 1900) >

See also

DDS_Property_t (p. 1900)

6.146 DDS_ProtocolVersion_t Struct Reference

<<*extension*>> (p. 1012) Type used to represent the version of the RTPS protocol.

Data Fields

- **DDS_Octet major**
Major protocol version number.
- **DDS_Octet minor**
Minor protocol version number.

6.146.1 Detailed Description

<<*extension*>> (p. 1012) Type used to represent the version of the RTPS protocol.

6.146.2 Field Documentation

6.146.2.1 major

DDS_Octet DDS_ProtocolVersion_t::major

Major protocol version number.

6.146.2.2 minor

DDS_Octet DDS_ProtocolVersion_t::minor

Minor protocol version number.

6.147 DDS_PublicationBuiltinTopicData Struct Reference

Entry created when a **DDS_DataWriter** (p. 485) is discovered in association with its Publisher.

Data Fields

- **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 **DDS_Topic** (p. 175).*
- char * **type_name**
*Name of the type attached to the **DDS_Topic** (p. 175).*
- 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. 1012) Type code information of the corresponding Topic
- **DDS_BuiltinTopicKey_t publisher_key**
<<extension>> (p. 1012) DCPS key of the publisher to which the DataWriter belongs
- struct **DDS_PropertyQosPolicy property**
<<extension>> (p. 1012) Properties of the corresponding DataWriter.
- struct **DDS_LocatorSeq unicast_locators**
<<extension>> (p. 1012) 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. 1012) Virtual GUID associated to the DataWriter.
- struct **DDS_ServiceQosPolicy service**
<<extension>> (p. 1012) Policy of the corresponding DataWriter.
- **DDS_ProtocolVersion_t rtps_protocol_version**
<<extension>> (p. 1012) Version number of the RTPS wire protocol used.
- struct **DDS_VendorId_t rtps_vendor_id**
<<extension>> (p. 1012) ID of vendor implementing the RTPS wire protocol.
- struct **DDS_ProductVersion_t product_version**
<<extension>> (p. 1012) This is a vendor specific parameter. It gives the current version for rti-dds.
- struct **DDS_LocatorFilterQosPolicy locator_filter**
<<extension>> (p. 1012) Policy of the corresponding DataWriter
- **DDS_Boolean disable_positive_acks**
<<extension>> (p. 1012) This is a vendor specific parameter. Determines whether matching DataReaders send positive acknowledgements for reliability.
- struct **DDS_EntityNameQosPolicy publication_name**
<<extension>> (p. 1012) The publication name and role name.
- struct **DDS_EndpointTrustProtectionInfo trust_protection_info**
<<extension>> (p. 1012) Trust Plugins protection information associated with the discovered DataWriter.
- struct **DDS_EndpointTrustAlgorithmInfo trust_algorithm_info**
<<extension>> (p. 1012) Trust Plugins algorithms associated with the discovered DataWriter.
- **DDS_BuiltinTopicKey_t related_datareader_key**
<<extension>> (p. 1012) Key of the related DataReader that is created as a part of the Remote Procedure Call (RPC) or Request-Reply endpoints.

6.147.1 Detailed Description

Entry created when a **DDS_DataWriter** (p. 485) is discovered in association with its Publisher.

Data associated with the built-in topic **DDS_PUBLICATION_TOPIC_NAME** (p. 1096). It contains QoS policies and additional information that apply to the remote **DDS_DataWriter** (p. 485) the related **DDS_Publisher** (p. 443).

See also

DDS_PUBLICATION_TOPIC_NAME (p. 1096)

DDS_PublicationBuiltinTopicDataDataReader (p. 1095)

6.147.2 Field Documentation

6.147.2.1 key

`DDS_BuiltinTopicKey_t DDS_PublicationBuiltinTopicData::key`

DCPS key to distinguish entries.

6.147.2.2 participant_key

`DDS_BuiltinTopicKey_t DDS_PublicationBuiltinTopicData::participant_key`

DCPS key of the participant to which the DataWriter belongs.

6.147.2.3 topic_name

`char* DDS_PublicationBuiltinTopicData::topic_name`

Name of the related **DDS_Topic** (p. 175).

The length of this string is limited to 255 characters.

The memory for this field is managed as described in **String Conventions** (p. 1527).

See also

String Conventions (p. 1527)

6.147.2.4 type_name

`char* DDS_PublicationBuiltinTopicData::type_name`

Name of the type attached to the **DDS_Topic** (p. 175).

The length of this string is limited to 255 characters.

The memory for this field is managed as described in **String Conventions** (p. 1527).

See also

String Conventions (p. 1527)

6.147.2.5 durability

```
struct DDS_DurabilityQosPolicy DDS_PublicationBuiltinTopicData::durability
```

durability policy of the corresponding DataWriter

6.147.2.6 durability_service

```
struct DDS_DurabilityServiceQosPolicy DDS_PublicationBuiltinTopicData::durability_service
```

durability_service policy of the corresponding DataWriter

6.147.2.7 deadline

```
struct DDS_DeadlineQosPolicy DDS_PublicationBuiltinTopicData::deadline
```

Policy of the corresponding DataWriter.

6.147.2.8 latency_budget

```
struct DDS_LatencyBudgetQosPolicy DDS_PublicationBuiltinTopicData::latency_budget
```

Policy of the corresponding DataWriter.

6.147.2.9 liveliness

```
struct DDS_LivelinessQosPolicy DDS_PublicationBuiltinTopicData::liveliness
```

Policy of the corresponding DataWriter.

6.147.2.10 reliability

```
struct DDS_ReliabilityQosPolicy DDS_PublicationBuiltinTopicData::reliability
```

Policy of the corresponding DataWriter.

6.147.2.11 lifespan

```
struct DDS_LifespanQosPolicy DDS_PublicationBuiltinTopicData::lifespan
```

Policy of the corresponding DataWriter.

6.147.2.12 user_data

```
struct DDS_UserDataQosPolicy DDS_PublicationBuiltinTopicData::user_data
```

Policy of the corresponding DataWriter.

6.147.2.13 ownership

```
struct DDS_OwnershipQosPolicy DDS_PublicationBuiltinTopicData::ownership
```

Policy of the corresponding DataWriter.

6.147.2.14 ownership_strength

```
struct DDS_OwnershipStrengthQosPolicy DDS_PublicationBuiltinTopicData::ownership_strength
```

Policy of the corresponding DataWriter.

6.147.2.15 destination_order

```
struct DDS_DestinationOrderQosPolicy DDS_PublicationBuiltinTopicData::destination_order
```

Policy of the corresponding DataWriter.

Warning

Only the field **DDS_DestinationOrderQosPolicy::kind** (p. 1688) is propagated during discovery. The other fields always contain their default values.

6.147.2.16 presentation

```
struct DDS_PresentationQosPolicy DDS_PublicationBuiltinTopicData::presentation
```

Policy of the Publisher to which the DataWriter belongs.

6.147.2.17 partition

```
struct DDS_PartitionQosPolicy DDS_PublicationBuiltinTopicData::partition
```

Policy of the Publisher to which the DataWriter belongs.

6.147.2.18 topic_data

```
struct DDS_TopicDataQosPolicy DDS_PublicationBuiltinTopicData::topic_data
```

Policy of the related Topic.

6.147.2.19 group_data

```
struct DDS_GroupDataQosPolicy DDS_PublicationBuiltinTopicData::group_data
```

Policy of the Publisher to which the DataWriter belongs.

6.147.2.20 representation

```
struct DDS_DataRepresentationQosPolicy DDS_PublicationBuiltinTopicData::representation
```

Data representation policy of the corresponding DataWriter.

6.147.2.21 data_tags

```
DDS_DataTagQosPolicy DDS_PublicationBuiltinTopicData::data_tags
```

Tags of the corresponding DataWriter.

6.147.2.22 type_code

```
struct DDS_TypeCode* DDS_PublicationBuiltinTopicData::type_code
```

<<*extension*>> (p. 1012) Type code information of the corresponding Topic

6.147.2.23 publisher_key

```
DDS_BuiltinTopicKey_t DDS_PublicationBuiltinTopicData::publisher_key
```

<<*extension*>> (p. 1012) DCPS key of the publisher to which the DataWriter belongs

6.147.2.24 property

```
struct DDS_PropertyQosPolicy DDS_PublicationBuiltinTopicData::property
```

<<*extension*>> (p. 1012) Properties of the corresponding DataWriter.

6.147.2.25 unicast_locators

```
struct DDS_LocatorSeq DDS_PublicationBuiltinTopicData::unicast_locators
```

<<*extension*>> (p. 1012) Custom unicast locators that the endpoint can specify. The default locators will be used if this is not specified.

6.147.2.26 virtual_guid

```
struct DDS_GUID_t DDS_PublicationBuiltinTopicData::virtual_guid
```

<<*extension*>> (p. 1012) Virtual GUID associated to the DataWriter.

See also

DDS_GUID_t (p. 1804)

6.147.2.27 service

```
struct DDS_ServiceQosPolicy DDS_PublicationBuiltinTopicData::service
```

<<*extension*>> (p. 1012) Policy of the corresponding DataWriter.

6.147.2.28 rtps_protocol_version

```
DDS_ProtocolVersion_t DDS_PublicationBuiltinTopicData::rtps_protocol_version
```

<<*extension*>> (p. 1012) Version number of the RTPS wire protocol used.

6.147.2.29 rtps_vendor_id

```
struct DDS_VendorId_t DDS_PublicationBuiltinTopicData::rtps_vendor_id
```

<<*extension*>> (p. 1012) ID of vendor implementing the RTPS wire protocol.

6.147.2.30 product_version

```
struct DDS_ProductVersion_t DDS_PublicationBuiltinTopicData::product_version
```

<<*extension*>> (p. 1012) This is a vendor specific parameter. It gives the current version for rti-dds.

6.147.2.31 locator_filter

```
struct DDS_LocatorFilterQosPolicy DDS_PublicationBuiltinTopicData::locator_filter
```

<<*extension*>> (p. 1012) Policy of the corresponding DataWriter

Related to **DDS_MultiChannelQosPolicy** (p. 1856).

6.147.2.32 disable_positive_acks

```
DDS_Boolean DDS_PublicationBuiltinTopicData::disable_positive_acks
```

<<*extension*>> (p. 1012) This is a vendor specific parameter. Determines whether matching DataReaders send positive acknowledgements for reliability.

6.147.2.33 publication_name

```
struct DDS_EntityNameQosPolicy DDS_PublicationBuiltinTopicData::publication_name
```

<<**extension**>> (p. 1012) The publication name and role name.

This member contains the name and the role name of the discovered publication.

6.147.2.34 trust_protection_info

```
struct DDS_EndpointTrustProtectionInfo DDS_PublicationBuiltinTopicData::trust_protection_info
```

<<**extension**>> (p. 1012) 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*.

6.147.2.35 trust_algorithm_info

```
struct DDS_EndpointTrustAlgorithmInfo DDS_PublicationBuiltinTopicData::trust_algorithm_info
```

<<**extension**>> (p. 1012) 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*.

6.147.2.36 related_datareader_key

```
DDS_BuiltinTopicKey_t DDS_PublicationBuiltinTopicData::related_datareader_key
```

<<**extension**>> (p. 1012) 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. 1111) only when the discovered DataWriter is a part of Remote Procedure Call (RPC) or Request-Reply endpoints.

6.148 DDS_PublicationBuiltinTopicDataSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_PublicationBuiltinTopicData** (p. 1905) > .

6.148.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_PublicationBuiltinTopicData** (p. 1905) > .

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_PublicationBuiltinTopicData (p. 1905)

6.149 DDS_PublicationBuiltinTopicDataTypeSupport Struct Reference

Instantiates **TypeSupport** < **DDS_PublicationBuiltinTopicData** (p. 1905) > .

6.149.1 Detailed Description

Instantiates **TypeSupport** < **DDS_PublicationBuiltinTopicData** (p. 1905) > .

Instantiates:

<<*generic*>> (p. 1013) **FooTypeSupport** (p. 2114)

See also

DDS_PublicationBuiltinTopicData (p. 1905)

6.150 DDS_PublicationMatchedStatus Struct Reference

DDS_PUBLICATION_MATCHED_STATUS (p. 1232)

Data Fields

- **DDS_Long total_count**

*The total cumulative number of times that this **DDS_DataWriter** (p. 485) discovered a "match" with a **DDS_DataReader** (p. 615).*

- **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 **DDS_DataWriter** (p. 485) is matched.*

- **DDS_Long current_count_peak**

*<<extension>> (p. 1012) Greatest number of DataReaders that matched this **DDS_DataWriter** (p. 485) 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 **DDS_DataReader** (p. 615) was the last to cause this DataWriter's status to change, using **DDS_DataWriter_get_matched_subscription_data** (p. 540).*

6.150.1 Detailed Description

DDS_PUBLICATION_MATCHED_STATUS (p. 1232)

A "match" happens when the **DDS_DataWriter** (p. 485) finds a **DDS_DataReader** (p. 615) with the same **DDS_Topic** (p. 175), same or compatible data type, and requested QoS that is compatible with that offered by the **DDS_DataWriter** (p. 485). (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 **DDS_DataWriter** (p. 485) will become "unmatched" from a remote **DDS_DataReader** (p. 615) when that **DDS_DataReader** (p. 615) goes away for any of the following reasons:

- The matched **DDS_DataReader** (p. 615)'s **DDS_DomainParticipant** (p. 77) 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 unmatched events, and the **DDS_PublicationMatchedStatus**↵
::**current_count_change** (p. 1916) can be used to determine the number of changes since the listener was called back or the status was checked.

6.150.2 Field Documentation

6.150.2.1 total_count

DDS_Long DDS_PublicationMatchedStatus::total_count

The total cumulative number of times that this **DDS_DataWriter** (p. 485) discovered a "match" with a **DDS_DataReader** (p. 615).

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. 1914).

6.150.2.2 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).

6.150.2.3 current_count

DDS_Long DDS_PublicationMatchedStatus::current_count

The current number of DataReaders with which this **DDS_DataWriter** (p. 485) 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. 1914).

6.150.2.4 current_count_peak

DDS_Long DDS_PublicationMatchedStatus::current_count_peak

<<**extension**>> (p. 1012) Greatest number of DataReaders that matched this **DDS_DataWriter** (p. 485) 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.)

6.150.2.5 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. 1914).

6.150.2.6 last_subscription_handle

DDS_InstanceHandle_t DDS_PublicationMatchedStatus::last_subscription_handle

This InstanceHandle can be used to look up which remote **DDS_DataReader** (p. 615) was the last to cause this DataWriter's status to change, using **DDS_DataWriter_get_matched_subscription_data** (p. 540).

If the DataReader no longer matches this DataWriter due to any of the reasons in **DDS_PublicationMatchedStatus** (p. 1914) 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 **DDS_DataWriter_get_matched_subscription_data** (p. 540) 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.

6.151 DDS_PublisherListener Struct Reference

<<*interface*>> (p. 1013) **DDS_Listener** (p. 1816) for **DDS_Publisher** (p. 443) status.

Data Fields

- struct **DDS_DataWriterListener** as_datawriterlistener
The superclass instance of this **DDS_PublisherListener** (p. 1917).

6.151.1 Detailed Description

<<*interface*>> (p. 1013) **DDS_Listener** (p. 1816) for **DDS_Publisher** (p. 443) status.

Entity:

DDS_Publisher (p. 443)

Status:

DDS_LIVELINESS_LOST_STATUS (p. 1231), **DDS_LivelinessLostStatus** (p. 1822);
DDS_OFFERED_DEADLINE_MISSED_STATUS (p. 1228), **DDS_OfferedDeadlineMissedStatus** (p. 1861);
DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 1229), **DDS_OfferedIncompatibleQosStatus** (p. 1862);
DDS_PUBLICATION_MATCHED_STATUS (p. 1232), **DDS_PublicationMatchedStatus** (p. 1914);
DDS_RELIABLE_READER_ACTIVITY_CHANGED_STATUS (p. 1234), **DDS_ReliableReaderActivityChangedStatus** (p. 1940);
DDS_RELIABLE_WRITER_CACHE_CHANGED_STATUS (p. 1234), **DDS_ReliableWriterCacheChangedStatus** (p. 1942)

See also

DDS_Listener (p. 1816)
Status Kinds (p. 1222)
Operations Allowed in Listener Callbacks (p. 1818)

6.151.2 Field Documentation

6.151.2.1 as_datawriterlistener

```
struct DDS_DataWriterListener DDS_PublisherListener::as_datawriterlistener
```

The superclass instance of this **DDS_PublisherListener** (p. 1917).

6.152 DDS_PublisherQos Struct Reference

QoS policies supported by a **DDS_Publisher** (p. 443) entity.

Data Fields

- struct **DDS_PresentationQosPolicy** **presentation**
Presentation policy, **PRESENTATION** (p. 1304).
- struct **DDS_PartitionQosPolicy** **partition**
<<**basic**>> (p. 783) Partition policy, **PARTITION** (p. 1303).
- struct **DDS_GroupDataQosPolicy** **group_data**
Group data policy, **GROUP_DATA** (p. 1293).
- struct **DDS_EntityFactoryQosPolicy** **entity_factory**
<<**basic**>> (p. 783) Entity factory policy, **ENTITY_FACTORY** (p. 1289).
- struct **DDS_AsynchronousPublisherQosPolicy** **asynchronous_publisher**
<<**extension**>> (p. 1012) Asynchronous publishing settings for the **DDS_Publisher** (p. 443) and all entities that are created by it.
- struct **DDS_EntityNameQosPolicy** **publisher_name**
<<**extension**>> (p. 1012) EntityName policy, **ENTITY_NAME** (p. 1290).

6.152.1 Detailed Description

QoS policies supported by a **DDS_Publisher** (p. 443) entity.

You must set certain members in a consistent manner:

length of **DDS_GroupDataQosPolicy::value** (p. 1804) <= **DDS_DomainParticipantResourceLimitsQosPolicy::publisher_group_data_max_length** (p. 1746)

length of **DDS_PartitionQosPolicy::name** (p. 1883) <= **DDS_DomainParticipantResourceLimitsQosPolicy::max_partitions** (p. 1748)

combined number of characters (including terminating 0) in **DDS_PartitionQosPolicy::name** (p. 1883) <= **DDS_DomainParticipantResourceLimitsQosPolicy::max_partition_cumulative_characters** (p. 1748)

If any of the above are not true, **DDS_Publisher_set_qos** (p. 462) and **DDS_Publisher_set_qos_with_profile** (p. 463) will fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222) and **DDS_DomainParticipant_create_publisher** (p. 103) will return NULL.

6.152.2 Field Documentation

6.152.2.1 presentation

```
struct DDS_PresentationQosPolicy DDS_PublisherQos::presentation
```

Presentation policy, **PRESENTATION** (p. 1304).

6.152.2.2 partition

```
struct DDS_PartitionQosPolicy DDS_PublisherQos::partition
```

<<*basic*>> (p. 783) Partition policy, **PARTITION** (p. 1303).

6.152.2.3 group_data

```
struct DDS_GroupDataQosPolicy DDS_PublisherQos::group_data
```

Group data policy, **GROUP_DATA** (p. 1293).

6.152.2.4 entity_factory

```
struct DDS_EntityFactoryQosPolicy DDS_PublisherQos::entity_factory
```

<<*basic*>> (p. 783) Entity factory policy, **ENTITY_FACTORY** (p. 1289).

6.152.2.5 asynchronous_publisher

```
struct DDS_AsynchronousPublisherQosPolicy DDS_PublisherQos::asynchronous_publisher
```

<<*extension*>> (p. 1012) Asynchronous publishing settings for the **DDS_Publisher** (p. 443) and all entities that are created by it.

6.152.2.6 publisher_name

```
struct DDS_EntityNameQosPolicy DDS_PublisherQos::publisher_name
```

<<*extension*>> (p. 1012) EntityName policy, **ENTITY_NAME** (p. 1290).

6.153 DDS_PublisherSeq Struct Reference

Declares IDL sequence < **DDS_Publisher** (p. 443) > .

6.153.1 Detailed Description

Declares IDL sequence < **DDS_Publisher** (p. 443) > .

See also

FooSeq (p. 2113)

6.154 DDS_PublishModeQosPolicy Struct Reference

<<*extension*>> (p. 795) Specifies how RTI Connexx sends application data on the network. This QoS policy can be used to tell RTI Connexx to use its *own* thread to send data, instead of the user thread.

Data Fields

- **DDS_PublishModeQosPolicyKind** kind
 <<*basic*>> (p. 783) <<*extension*>> (p. 795) *Publishing mode*
- char * **flow_controller_name**
 <<*extension*>> (p. 795) *Name of the associated flow controller*
- **DDS_Long** priority
 <<*extension*>> (p. 795) *Publication priority*

6.154.1 Detailed Description

<<**extension**>> (p. 795) Specifies how RTI Connexx sends application data on the network. This QoS policy can be used to tell RTI Connexx to use its *own* thread to send data, instead of the user thread.

The publishing mode of a **DDS_DataWriter** (p. 485) determines whether data is written synchronously in the context of the user thread when calling **FooDataWriter_write** (p. 496) or asynchronously in the context of a separate thread internal to the middleware.

Each **DDS_Publisher** (p. 443) spawns a single asynchronous publishing thread (**DDS_AsynchronousPublisherQosPolicy::thread** (p. 1540)) to serve all its asynchronous **DDS_DataWriter** (p. 485) instances.

See also

DDS_AsynchronousPublisherQosPolicy (p. 1539)

DDS_HistoryQosPolicy (p. 1805)

DDS_FlowController (p. 559)

Entity:

DDS_DataWriter (p. 485)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

6.154.2 Usage

The fastest way for RTI Connexx 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 **DDS_FlowController** (p. 559), 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 **DDS_DataWriter** (p. 485). 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. 496) 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. 2124) (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. 2124). 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 **DDS_FlowController** (p. 559)). The number of samples that will be queued is determined by the **DDS_HistoryQosPolicy** (p. 1805). When using **DDS_KEEP_LAST_HISTORY_QOS** (p. 1293), only the most recent **DDS_HistoryQosPolicy::depth** (p. 1807) 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.

6.154.3 Field Documentation

6.154.3.1 kind

DDS_PublishModeQosPolicyKind **DDS_PublishModeQosPolicy::kind**

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Publishing mode

[default] **DDS_SYNCHRONOUS_PUBLISH_MODE_QOS** (p. 1319)

Categories

Immutable (p. 914), **Flow Control** (p. 863)

6.154.3.2 flow_controller_name

char* **DDS_PublishModeQosPolicy::flow_controller_name**

<<**extension**>> (p. 795) Name of the associated flow controller

The following builtin values are supported:

- **DDS_DEFAULT_FLOW_CONTROLLER_NAME** (p. 564)
- **DDS_FIXED_RATE_FLOW_CONTROLLER_NAME** (p. 564)
- **DDS_ON_DEMAND_FLOW_CONTROLLER_NAME** (p. 565)

NULL value or zero-length string refers to **DDS_DEFAULT_FLOW_CONTROLLER_NAME** (p. 564).

Warning

The value for this field can be one of the constants above or a string allocated with **DDS_String_dup()** (p. 1528) 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. 1528), 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. 1528) are released when `writer_qos` is finalized.

More information about string handling can be found under **String Conventions** (p. 1527).

See also

DDS_DomainParticipant_create_flowcontroller (p. 124)

[default] **DDS_DEFAULT_FLOW_CONTROLLER_NAME** (p. 564)

Categories

Immutable (p. 914), **Flow Control** (p. 863)

6.154.3.3 priority

DDS_Long DDS_PublishModeQosPolicy::priority

<<*extension*>> (p. 795) Publication priority

A positive integer value designating the relative priority of the **DDS_DataWriter** (p. 485), used to determine the transmission order of pending writes.

Use of publication priorities requires the asynchronous publisher (**DDS_ASYNCHRONOUS_PUBLISH_MODE**↔**_QOS** (p. 1319)) with **DDS_FlowControllerProperty_t::scheduling_policy** (p. 1799) set to **DDS_HPF_FLOW**↔**CONTROLLER_SCHED_POLICY** (p. 561).

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. 1318), 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. 1318), then the channel will inherit the publication priority of the DataWriter.

If the publication priority of the DataWriter, and of any channel of a multi-channel DataWriter, are **DDS_PUBLICATION**↔**_PRIORITY_UNDEFINED** (p. 1318), 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. 1318), then the Data↔Writer 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. 2101) of the **FooDataWriter_write**↔**_w_params** (p. 501) function.

For dispose and unregister samples, use the **DDS_WriteParams_t** (p. 2101) of **FooDataWriter_dispose_w_params** (p. 504) and **FooDataWriter_unregister_instance_w_params** (p. 496).

[default] **DDS_PUBLICATION_PRIORITY_UNDEFINED** (p. 1318)

[range] [-1, MAX_INT]

Categories

Immutable (p. 914), **Flow Control** (p. 863)

6.155 DDS_QosPolicyCount Struct Reference

Type to hold a counter for a **DDS_QosPolicyId_t** (p. 1246).

Data Fields

- **DDS_QosPolicyId_t** `policy_id`
The QosPolicy ID.
- **DDS_Long** `count`
a counter

6.155.1 Detailed Description

Type to hold a counter for a **DDS_QosPolicyId_t** (p. 1246).

6.155.2 Field Documentation

6.155.2.1 `policy_id`

DDS_QosPolicyId_t `DDS_QosPolicyCount::policy_id`

The QosPolicy ID.

6.155.2.2 `count`

DDS_Long `DDS_QosPolicyCount::count`

a counter

6.156 DDS_QosPolicyCountSeq Struct Reference

Declares IDL sequence < **DDS_QosPolicyCount** (p. 1924) >

6.156.1 Detailed Description

Declares IDL sequence < **DDS_QosPolicyCount** (p. 1924) >

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_QosPolicyCount (p. 1924)

6.157 DDS_QosPrintFormat Struct Reference

A collection of attributes used to configure how a QoS appears when printed.

Data Fields

- **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**
Configures how much additional indent should be added to the string representation of a QoS.

6.157.1 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. 1245).

6.157.2 Field Documentation

6.157.2.1 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

6.157.2.2 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

6.157.2.3 indent

DDS_UnsignedLong DDS_QosPrintFormat::indent

Configures 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

6.158 DDS_QueryConditionParams Struct Reference

<<**extension**>> (p. 1012) Input parameters for **DDS_DataReader_create_querycondition_w_params** (p. 673)

Data Fields

- 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.

6.158.1 Detailed Description

<<**extension**>> (p. 1012) Input parameters for **DDS_DataReader_create_querycondition_w_params** (p. 673)

6.158.2 Field Documentation

6.158.2.1 as_readconditionparams

```
struct DDS_ReadConditionParams DDS_QueryConditionParams::as_readconditionparams
```

Read condition parameters.

6.158.2.2 query_expression

```
char* DDS_QueryConditionParams::query_expression
```

Expression for the query.

Cannot be NULL.

6.158.2.3 query_parameters

```
struct DDS_StringSeq DDS_QueryConditionParams::query_parameters
```

Parameters for the query expression.

6.159 DDS_ReadConditionParams Struct Reference

<<*extension*>> (p. 1012) Input parameters for **DDS_DataReader_create_readcondition_w_params** (p. 672)

Data Fields

- **DDS_SampleStateMask sample_states**
Sample state.
- **DDS_ViewStateMask view_states**
View state.
- **DDS_InstanceStateMask instance_states**
Instance state.
- **DDS_StreamKindMask stream_kinds**
Stream kind.

6.159.1 Detailed Description

<<*extension*>> (p. 1012) Input parameters for **DDS_DataReader_create_readcondition_w_params** (p. 672)

6.159.2 Field Documentation

6.159.2.1 sample_states

`DDS_SampleStateMask DDS_ReadConditionParams::sample_states`

Sample state.

Sample state of the data samples that are of interest.

6.159.2.2 view_states

`DDS_ViewStateMask DDS_ReadConditionParams::view_states`

View state.

View state of the data samples that are of interest.

6.159.2.3 instance_states

`DDS_InstanceStateMask DDS_ReadConditionParams::instance_states`

Instance state.

Instance state of the data samples that are of interest.

6.159.2.4 stream_kinds

`DDS_StreamKindMask DDS_ReadConditionParams::stream_kinds`

Stream kind.

Stream kind of the data samples that are of interest.

6.160 DDS_ReaderDataLifecycleQosPolicy Struct Reference

<<*spec*>> (p. 790) Controls how a DataReader manages the lifecycle of the data that it has received.

Data Fields

- struct **DDS_Duration_t autopurge_nowriter_samples_delay**
 <<**spec**>> (p. 790) Minimum duration for which the **DDS_DataReader** (p. 615) will maintain samples for an instance once its *instance_state* becomes **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 715).
- struct **DDS_Duration_t autopurge_disposed_samples_delay**
 <<**spec**>> (p. 790) Minimum duration for which the **DDS_DataReader** (p. 615) will maintain samples for an instance once its *instance_state* becomes **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 715).
- struct **DDS_Duration_t autopurge_disposed_instances_delay**
 <<**extension**>> (p. 795) Minimum duration for which the **DDS_DataReader** (p. 615) will maintain information about a received instance once its *instance_state* becomes **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 715) and there are no samples for the instance in the DataReader queue.
- struct **DDS_Duration_t autopurge_nowriter_instances_delay**
 <<**extension**>> (p. 795) Minimum duration for which the **DDS_DataReader** (p. 615) will maintain information about a received instance once its *instance_state* becomes **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 715) and there are no samples for the instance in the DataReader queue.

6.160.1 Detailed Description

<<**spec**>> (p. 790) 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 **DDS_DataReader** (p. 615) must retain information regarding instances that have the *instance_state* **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 715).

Note: This policy is not concerned with keeping reliable reader state or discovery information.

The **DDS_DataReader** (p. 615) 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. 1805) and **DDS_ResourceLimitsQosPolicy** (p. 1949).

The **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615) will have taken for that instance will have an *instance_state* of either **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 715) or **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 715) depending on whether or not the last writer that had ownership of the instance disposed it.

In the absence of **READER_DATA_LIFECYCLE** (p. 1320), this behavior could cause problems if the application forgets to take those samples. "Untaken" samples will prevent the **DDS_DataReader** (p. 615) from reclaiming the resources and they would remain in the **DDS_DataReader** (p. 615) indefinitely.

A DataReader can also reclaim all resources for instances that have an *instance_state* of **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 715) 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:

DDS_DataReader (p. 615)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **YES** (p. 1245)

6.160.2 Field Documentation

6.160.2.1 autopurge_nowriter_samples_delay

```
struct DDS_Duration_t DDS_ReaderDataLifecycleQosPolicy::autopurge_nowriter_samples_delay
```

<<*spec*>> (p. 790) Minimum duration for which the **DDS_DataReader** (p. 615) will maintain samples for an instance once its `instance_state` becomes **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 715).

At some point after this time elapses, the **DDS_DataReader** (p. 615) will purge all samples for the instance; any "un-taken" samples will also be dropped.

[default] **DDS_DURATION_INFINITE** (p. 1208)

[range] [1 nanosec, 1 year] or **DDS_DURATION_INFINITE** (p. 1208)

Categories

Immutable (p. 914), **Instance Lifecycle** (p. 867), **Sample Memory Management** (p. 902), **Data Availability** (p. 843)

6.160.2.2 autopurge_disposed_samples_delay

```
struct DDS_Duration_t DDS_ReaderDataLifecycleQosPolicy::autopurge_disposed_samples_delay
```

<<*spec*>> (p. 790) Minimum duration for which the **DDS_DataReader** (p. 615) will maintain samples for an instance once its `instance_state` becomes **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 715).

After this time elapses, the **DDS_DataReader** (p. 615) 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. 1208)

[range] [1 nanosec, 1 year] or **DDS_DURATION_INFINITE** (p. 1208)

Categories

Immutable (p. 914), **Instance Lifecycle** (p. 867), **Sample Memory Management** (p. 902), **Data Availability** (p. 843)

6.160.2.3 autopurge_disposed_instances_delay

```
struct DDS_Duration_t DDS_ReaderDataLifecycleQosPolicy::autopurge_disposed_instances_delay
```

<<**extension**>> (p. 795) Minimum duration for which the **DDS_DataReader** (p. 615) will maintain information about a received instance once its `instance_state` becomes **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 715) 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 **DDS_DataReader** (p. 615) 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. 1619).

The only currently supported values are **DDS_DURATION_ZERO** (p. 1209) and **DDS_DURATION_INFINITE** (p. 1208). A value of **DDS_DURATION_ZERO** (p. 1209) will purge an instance's state immediately after the instance state transitions to **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 715), as long as all samples, including the dispose sample, associated with that instance have been 'taken'.

[default] **DDS_DURATION_INFINITE** (p. 1208)

Categories

Immutable (p. 914), **Instance Lifecycle** (p. 867), **Sample Memory Management** (p. 902), **Data Availability** (p. 843)

6.160.2.4 autopurge_nowriter_instances_delay

```
struct DDS_Duration_t DDS_ReaderDataLifecycleQosPolicy::autopurge_nowriter_instances_delay
```

<<**extension**>> (p. 795) Minimum duration for which the **DDS_DataReader** (p. 615) will maintain information about a received instance once its `instance_state` becomes **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 715) 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. 715) `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 **DDS_DataReader** (p. 615) 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. 1619).

The only currently supported values are **DDS_DURATION_ZERO** (p. 1209) and **DDS_DURATION_INFINITE** (p. 1208). A value of **DDS_DURATION_ZERO** (p. 1209) will purge an instance's state immediately after the instance state transitions to **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 715), as long as all samples, including the no_←writers sample, associated with that instance have been 'taken'.

[default] **DDS_DURATION_ZERO** (p. 1209)

Categories

Immutable (p. 914), **Instance Lifecycle** (p. 867), **Sample Memory Management** (p. 902), **Data Availability** (p. 843)

6.161 DDS_ReceiverPoolQosPolicy Struct Reference

<<*extension*>> (p. 795) Configures threads used by RTI Connex to receive and process data from transports (for example, UDP sockets).

Data Fields

- struct **DDS_ThreadSettings_t** **thread**
 <<*extension*>> (p. 795) *Receiver pool thread(s).*
- **DDS_Long** **buffer_size**
 <<*extension*>> (p. 795) *The receive buffer size in bytes.*
- **DDS_Long** **buffer_alignment**
 [DEPRECATED] (p. 965) <<*extension*>> (p. 795) *The receive buffer alignment.*

6.161.1 Detailed Description

<<*extension*>> (p. 795) Configures threads used by RTI Connex to receive and process data from transports (for example, UDP sockets).

This QoS policy is an extension to the DDS standard.

Entity:

DDS_DomainParticipant (p. 77)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

See also

Controlling CPU Core Affinity for RTI Threads (p. 1237)

6.161.2 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 Connex to use a separate thread to process the data for a **DDS_DataReader** (p. 615), set a unique port for the **DDS_TransportUnicastQosPolicy** (p. 2065) or **DDS_TransportMulticastQosPolicy** (p. 2058) for the **DDS_DataReader** (p. 615).

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 **DDS_DomainParticipant** (p. 77) 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.

6.161.3 Field Documentation

6.161.3.1 thread

```
struct DDS_ThreadSettings_t DDS_ReceiverPoolQosPolicy::thread
```

<<**extension**>> (p. 795) 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. 1237) | `DDS_THREAD_SETTINGS_STUDIO` (p. 1237)

Categories

Immutable (p. 914), **Concurrency** (p. 839)

6.161.3.2 buffer_size

```
DDS_Long DDS_ReceiverPoolQosPolicy::buffer_size
```

<<**extension**>> (p. 795) 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. 2124) across *all* of the transports being used that are not doing zero-copy.

By default (**DDS_LENGTH_AUTO** (p. 1321)): the size is equal to the maximum **NDDS_Transport_Property_t::message_size_max** (p. 2124) 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. 2124) 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. 2124).

[default] **DDS_LENGTH_AUTO** (p. 1321)

[range] [1, 1 GB] or **DDS_LENGTH_AUTO** (p. 1321)

Categories

Immutable (p. 914), **Middleware Memory Management** (p. 878)

6.161.3.3 buffer_alignment

DDS_Long DDS_ReceiverPoolQosPolicy::buffer_alignment

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) 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. 914), **Middleware Memory Management** (p. 878)

6.162 DDS_ReliabilityQosPolicy Struct Reference

<<*extension*>> (p. 795) <<*spec*>> (p. 790) Indicates the level of reliability offered/requested by RTI Connext.

Data Fields

- **DDS_ReliabilityQosPolicyKind** **kind**
 <<*basic*>> (p. 783) <<*spec*>> (p. 790) *Kind of reliability.*
- struct **DDS_Duration_t** **max_blocking_time**
 <<*basic*>> (p. 783) <<*spec*>> (p. 790) *The maximum time a DataWriter can block on a write() call.*
- **DDS_ReliabilityQosPolicyAcknowledgmentModeKind** **acknowledgment_kind**
 <<*extension*>> (p. 795) *Kind of reliable acknowledgment*
- **DDS_InstanceStateConsistencyKind** **instance_state_consistency_kind**
 <<*extension*>> (p. 795) *Whether instance state consistency is enabled*

6.162.1 Detailed Description

<<*extension*>> (p. 795) <<*spec*>> (p. 790) Indicates the level of reliability offered/requested by RTI Connext.

Entity:

DDS_Topic (p. 175), **DDS_DataReader** (p. 615), **DDS_DataWriter** (p. 485)

Status:

DDS_OFFERED_INCOMPATIBLE_QOS_STATUS (p. 1229), **DDS_REQUESTED_INCOMPATIBLE_QOS_↵
STATUS** (p. 1229)

Properties:

RxO (p. 1245) = YES
Changeable (p. 1245) = **UNTIL ENABLE** (p. 1245) (the **instance_state_consistency_kind** is **Changeable** (p. 1245)
 = **NO** (p. 1245))

6.162.2 Usage

This policy indicates the level of reliability requested by a **DDS_DataReader** (p. 615) or offered by a **DDS_DataWriter** (p. 485).

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. 1323) 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. 1805), `DDS_DataWriterProtocolQosPolicy` (p. 1649), `DDS_DataReaderProtocolQosPolicy` (p. 1596), and `DDS_ResourceLimitsQosPolicy` (p. 1949).

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. 1323) < `DDS_RELIABLE_RELIABILITY_QOS` (p. 1323). A `DDS_DataWriter` (p. 485) 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. 1325) policy. In case the reliability kind is set to `DDS_RELIABLE_RELIABILITY_QOS` (p. 1323) the write operation on the `DDS_DataWriter` (p. 485) 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. 1325) to be exceeded. Under these circumstances, the `RELIABILITY` (p. 1321) `max_blocking_time` configures the maximum duration the write operation may block.

If the `DDS_ReliabilityQosPolicy::kind` (p. 1939) is set to `DDS_RELIABLE_RELIABILITY_QOS` (p. 1323), data samples originating from a single `DDS_DataWriter` (p. 485) cannot be made available to the `DDS_DataReader` (p. 615) 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 `DDS_DataWriter` (p. 485) history before it is accessible by the `DDS_DataReader` (p. 615).

If the `DDS_ReliabilityQosPolicy::kind` (p. 1939) is set to `DDS_BEST_EFFORT_RELIABILITY_QOS` (p. 1323), 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 `DDS_DataReader` (p. 615) history in the same order they originated in the `DDS_DataWriter` (p. 485). In other words, the `DDS_DataReader` (p. 615) 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. 1805)

`DDS_ResourceLimitsQosPolicy` (p. 1949)

6.162.3 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. 1939) are considered ordered such that `DDS_BEST_EFFORT_RELIABILITY_QOS` (p. 1323) < `DDS_RELIABLE_RELIABILITY_QOS` (p. 1323).
- The offered `acknowledgment_kind` = `DDS_PROTOCOL_ACKNOWLEDGMENT_MODE` (p. 1323) and requested `acknowledgment_kind` = `DDS_PROTOCOL_ACKNOWLEDGMENT_MODE` (p. 1323) OR offered `acknowledgment_kind` != `DDS_PROTOCOL_ACKNOWLEDGMENT_MODE` (p. 1323).
- the inequality `offered instance_state_consistency_kind >= requested instance_state_consistency_kind` evaluates to 'TRUE'. For the purposes of this inequality, the values of `DDS_ReliabilityQosPolicy::instance_state_consistency_kind` (p. 1940) are considered ordered such that `DDS_NO_RECOVER_INSTANCE_STATE_CONSISTENCY` (p. 1324) < `DDS_RECOVER_INSTANCE_STATE_CONSISTENCY` (p. 1324).

6.162.4 Field Documentation

6.162.4.1 kind

`DDS_ReliabilityQosPolicyKind DDS_ReliabilityQosPolicy::kind`

<<*basic*>> (p. 783) <<*spec*>> (p. 790) Kind of reliability.

[default] `DDS_BEST_EFFORT_RELIABILITY_QOS` (p. 1323) for `DDS_DataReader` (p. 615) and `DDS_Topic` (p. 175), `DDS_RELIABLE_RELIABILITY_QOS` (p. 1323) for `DDS_DataWriter` (p. 485)

Categories

Immutable (p. 914), **Matching** (p. 963), **Discovery Metadata** (p. 958), **Reliability** (p. 895)

6.162.4.2 max_blocking_time

`struct DDS_Duration_t DDS_ReliabilityQosPolicy::max_blocking_time`

<<*basic*>> (p. 783) <<*spec*>> (p. 790) 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. 1208)

See also

DDS_ResourceLimitsQosPolicy (p. 1949)

Categories

Immutable (p. 914), **Discovery Metadata** (p. 958), **Reliability** (p. 895), **Flow Control** (p. 863)

6.162.4.3 acknowledgment_kind

DDS_ReliabilityQosPolicyAcknowledgmentModeKind DDS_ReliabilityQosPolicy::acknowledgment_kind

<<**extension**>> (p. 795) Kind of reliable acknowledgment

This setting applies only to the case where **DDS_ReliabilityQosPolicy::kind** (p. 1939) = **DDS_RELIABLE_**↵
RELIABILITY_QOS (p. 1323).

Sets the kind acknowledgments supported by a **DDS_DataWriter** (p. 485) and sent by **DDS_DataReader** (p. 615).

[default] **DDS_PROTOCOL_ACKNOWLEDGMENT_MODE** (p. 1323)

Categories

Immutable (p. 914), **Matching** (p. 963), **Discovery Metadata** (p. 958), **Reliability** (p. 895), **Application Acknowledgement** (p. 835)

6.162.4.4 instance_state_consistency_kind

DDS_InstanceStateConsistencyKind DDS_ReliabilityQosPolicy::instance_state_consistency_kind

<<**extension**>> (p. 795) 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. 1324) for **DDS_DataReader** (p. 615), **DDS_**↵
_RECOVER_INSTANCE_STATE_CONSISTENCY (p. 1324) for **DDS_DataWriter** (p. 485)

Categories

Immutable (p. 914), **Matching** (p. 963), **Discovery Metadata** (p. 958), **Reliability** (p. 895), **Instance Lifecycle** (p. 867)

6.163 DDS_ReliableReaderActivityChangedStatus Struct Reference

<<**extension**>> (p. 1012) Describes the activity (i.e. are acknowledgements forthcoming) of reliable readers matched to a reliable writer.

Data Fields

- **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.

6.163.1 Detailed Description

<<**extension**>> (p. 1012) Describes the activity (i.e. are acknowledgements forthcoming) of reliable readers matched to a reliable writer.

Entity:

DDS_DataWriter (p. 485)

Listener:

DDS_DataWriterListener (p. 1644)

This status is the reciprocal status to the **DDS_LivelinessChangedStatus** (p. 1820) on the reader. It is different than the **DDS_LivelinessLostStatus** (p. 1822) 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.

6.163.2 Field Documentation

6.163.2.1 active_count

DDS_Long DDS_ReliableReaderActivityChangedStatus::active_count

The current number of reliable readers currently matched with this reliable writer.

6.163.2.2 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. 1966) times, each heartbeat having been separated from the previous by the current heartbeat period.

6.163.2.3 active_count_change

DDS_Long DDS_ReliableReaderActivityChangedStatus::active_count_change

The most recent change in the number of active remote reliable readers.

6.163.2.4 inactive_count_change

DDS_Long DDS_ReliableReaderActivityChangedStatus::inactive_count_change

The most recent change in the number of inactive remote reliable readers.

6.163.2.5 last_instance_handle

DDS_InstanceHandle_t DDS_ReliableReaderActivityChangedStatus::last_instance_handle

The instance handle of the last reliable remote reader to be determined inactive.

6.164 DDS_ReliableWriterCacheChangedStatus Struct Reference

<<*extension*>> (p. 1012) A summary of the state of a data writer's cache of unacknowledged samples written.

Data Fields

- 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.

6.164.1 Detailed Description

<<**extension**>> (p. 1012) A summary of the state of a data writer's cache of unacknowledged samples written.

Entity:

DDS_DataWriter (p. 485)

Listener:

DDS_DataWriterListener (p. 1644)

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.

6.164.2 Field Documentation

6.164.2.1 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.

6.164.2.2 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. 1973) and **DDS_RtpsReliableWriterProtocol_t::max_send_window_size** (p. 1973) are not
DDS_LENGTH_UNLIMITED (p. 1326)).

Otherwise, applies when the number of unacknowledged samples has reached the send window limit.

6.164.2.3 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. 1961) field of the **DDS_RtpsReliableWriterProtocol_t** (p. 1960) structure.

6.164.2.4 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. 1962) field of the **DDS_RtpsReliableWriterProtocol_t** (p. 1960) structure.

6.164.2.5 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.

6.164.2.6 unacknowledged_sample_count_peak

DDS_Long DDS_ReliableWriterCacheChangedStatus::unacknowledged_sample_count_peak

The highest value that unacknowledged_sample_count has reached until now.

6.164.2.7 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. 1293) policy.

6.165 DDS_ReliableWriterCacheEventCount Struct Reference

<<**extension**>> (p. 1012) The number of times the number of unacknowledged samples in the cache of a reliable writer hit a certain well-defined threshold.

Data Fields

- **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.

6.165.1 Detailed Description

<<**extension**>> (p. 1012) 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. 1942).

See also

DDS_ReliableWriterCacheChangedStatus (p. 1942)

6.165.2 Field Documentation

6.165.2.1 total_count

DDS_Long DDS_ReliableWriterCacheEventCount::total_count

The total number of times the event has occurred.

6.165.2.2 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.

6.166 DDS_RequestedDeadlineMissedStatus Struct Reference

DDS_REQUESTED_DEADLINE_MISSED_STATUS (p. 1229)

Data Fields

- **DDS_Long total_count**
*Total cumulative count of the missed deadlines detected for any instance read by the **DDS_DataReader** (p. 615).*
- **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 **DDS_DataReader** (p. 615) for which a requested deadline was missed.*

6.166.1 Detailed Description

DDS_REQUESTED_DEADLINE_MISSED_STATUS (p. 1229)

Examples

HelloWorld_subscriber.c.

6.166.2 Field Documentation

6.166.2.1 total_count

`DDS_Long DDS_RequestedDeadlineMissedStatus::total_count`

Total cumulative count of the missed deadlines detected for any instance read by the **DDS_DataReader** (p. 615).

6.166.2.2 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.

6.166.2.3 last_instance_handle

`DDS_InstanceHandle_t DDS_RequestedDeadlineMissedStatus::last_instance_handle`

Handle to the last instance in the **DDS_DataReader** (p. 615) for which a requested deadline was missed.

6.167 DDS_RequestedIncompatibleQoSStatus Struct Reference

DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS (p. 1229)

Data Fields

- **DDS_Long total_count**

*Total cumulative count of how many times the concerned **DDS_DataReader** (p. 615) discovered a **DDS_DataWriter** (p. 485) for the same **DDS_Topic** (p. 175) with an offered QoS that is incompatible with that requested by the **DDS_DataReader** (p. 615).*

- **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. 1246) 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 **DDS_DataReader** (p. 615) discovered a **DDS_DataWriter** (p. 485) for the same **DDS_Topic** (p. 175) with an offered QoS that is incompatible with that requested by the **DDS_DataReader** (p. 615).*

6.167.1 Detailed Description

DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS (p. 1229)

See also

DURABILITY (p. 1285)

PRESENTATION (p. 1304)

RELIABILITY (p. 1321)

OWNERSHIP (p. 1301)

LIVELINESS (p. 1296)

DEADLINE (p. 1272)

LATENCY_BUDGET (p. 1294)

DESTINATION_ORDER (p. 1273)

Examples

HelloWorld_subscriber.c.

6.167.2 Field Documentation

6.167.2.1 total_count

DDS_Long DDS_RequestedIncompatibleQosStatus::total_count

Total cumulative count of how many times the concerned **DDS_DataReader** (p. 615) discovered a **DDS_DataWriter** (p. 485) for the same **DDS_Topic** (p. 175) with an offered QoS that is incompatible with that requested by the **DDS_↔DataReader** (p. 615).

6.167.2.2 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.

6.167.2.3 last_policy_id

```
DDS_QosPolicyId_t DDS_RequestedIncompatibleQosStatus::last_policy_id
```

The **DDS_QosPolicyId_t** (p. 1246) of one of the policies that was found to be incompatible the last time an incompatibility was detected.

6.167.2.4 policies

```
struct DDS_QosPolicyCountSeq DDS_RequestedIncompatibleQosStatus::policies
```

A list containing, for each policy, the total number of times that the concerned **DDS_DataReader** (p. 615) discovered a **DDS_DataWriter** (p. 485) for the same **DDS_Topic** (p. 175) with an offered QoS that is incompatible with that requested by the **DDS_DataReader** (p. 615).

6.168 DDS_ResourceLimitsQosPolicy Struct Reference

<<**spec**>> (p. 790) 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.

Data Fields

- **DDS_Long max_samples**
 <<**basic**>> (p. 783) <<**spec**>> (p. 790) Represents the maximum samples the middleware can store for any one **DDS_DataWriter** (p. 485) (or **DDS_DataReader** (p. 615)).
- **DDS_Long max_instances**
 <<**basic**>> (p. 783) <<**spec**>> (p. 790) Represents the maximum number of instances a **DDS_DataWriter** (p. 485) (or **DDS_DataReader** (p. 615)) can manage.
- **DDS_Long max_samples_per_instance**
 <<**spec**>> (p. 790) Represents the maximum number of samples of any one instance a **DDS_DataWriter** (p. 485) (or **DDS_DataReader** (p. 615)) can manage.
- **DDS_Long initial_samples**
 <<**basic**>> (p. 783) <<**extension**>> (p. 795) Represents the initial samples the middleware will store for any one **DDS_DataWriter** (p. 485) (or **DDS_DataReader** (p. 615)).
- **DDS_Long initial_instances**
 <<**basic**>> (p. 783) <<**extension**>> (p. 795) Represents the initial number of instances a **DDS_DataWriter** (p. 485) (or **DDS_DataReader** (p. 615)) will manage.
- **DDS_Long instance_hash_buckets**
 [DEPRECATED] (p. 965) <<**extension**>> (p. 795) Number of hash buckets for instances.

6.168.1 Detailed Description

<<*spec*>> (p. 790) 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:

DDS_Topic (p. 175), **DDS_DataReader** (p. 615), **DDS_DataWriter** (p. 485)

Status:

DDS_SAMPLE_REJECTED_STATUS (p. 1230), **DDS_SampleRejectedStatus** (p. 1997)

Properties:

RxO (p. 1245) = NO

Changeable (p. 1245) = **UNTIL ENABLE** (p. 1245)

6.168.2 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. 1759)), this QoS policy determines the actual maximum queue size when the **DDS_HistoryQosPolicy** (p. 1805) is set to **DDS_KEEP_ALL_HISTORY_QOS** (p. 1293).

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 **DDS_DataWriter** (p. 485) objects are communicating samples faster than they are ultimately taken by the **DDS_DataReader** (p. 615) objects, the middleware will eventually hit against some of the QoS-imposed resource limits. Note that this may occur when just a single **DDS_DataReader** (p. 615) cannot keep up with its corresponding **DDS_DataWriter** (p. 485). The behavior in this case depends on the setting for the **RELIABILITY** (p. 1321). If reliability is **DDS_BEST_EFFORT_RELIABILITY_QOS** (p. 1323), then RTI Connext is allowed to drop samples. If the reliability is **DDS_RELIABLE_RELIABILITY_QOS** (p. 1323), RTI Connext will block the **DDS_DataWriter** (p. 485) or discard the sample at the **DDS_DataReader** (p. 615) in order not to lose existing samples.

The constant **DDS_LENGTH_UNLIMITED** (p. 1326) may be used to indicate the absence of a particular limit. For example setting **DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1952) to **DDS_LENGTH_UNLIMITED** (p. 1326) will cause RTI Connext not to enforce this particular limit.

If these resource limits are not set sufficiently, under certain circumstances the **DDS_DataWriter** (p. 485) may block on a write() call even though the **DDS_HistoryQosPolicy** (p. 1805) is **DDS_KEEP_LAST_HISTORY_QOS** (p. 1293). To guarantee the writer does not block for **DDS_KEEP_LAST_HISTORY_QOS** (p. 1293), make sure the resource limits are set such that:

```
max_samples >= max_instances * max_samples_per_instance
```

See also

DDS_ReliabilityQosPolicy (p. 1936)

DDS_HistoryQosPolicy (p. 1805)

6.168.3 Consistency

The setting of **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1951) must be consistent with **DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1952). For these two values to be consistent, it must be true that **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1951) $\geq$ **DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1952). As described above, this limit will not be enforced if **DDS_ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1952) is set to **DDS_LENGTH_UNLIMITED** (p. 1326).

The setting of **RESOURCE_LIMITS** (p. 1325) **max_samples_per_instance** must be consistent with the **HISTORY** (p. 1292) **depth**. For these two QoS to be consistent, it must be true that *depth* $\leq$ *max_samples_per_instance*.

See also

DDS_HistoryQosPolicy (p. 1805)

6.168.4 Field Documentation

6.168.4.1 max_samples

DDS_Long **DDS_ResourceLimitsQosPolicy::max_samples**

<<basic>> (p. 783) **<<spec>>** (p. 790) Represents the maximum samples the middleware can store for any one **DDS_DataWriter** (p. 485) (or **DDS_DataReader** (p. 615)).

Specifies the maximum number of data samples a **DDS_DataWriter** (p. 485) (or **DDS_DataReader** (p. 615)) 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. 1326).

When batching is enabled, the maximum number of data samples a **DDS_DataWriter** (p. 485) can manage will also be limited by **DDS_DataWriterResourceLimitsQosPolicy::max_batches** (p. 1677).

[default] **DDS_LENGTH_UNLIMITED** (p. 1326)

[range] [1, 100 million] or **DDS_LENGTH_UNLIMITED** (p. 1326), $\geq$ **initial_samples**, $\geq$ **max_samples_per_instance**, $\geq$ **DDS_DataReaderResourceLimitsQosPolicy::max_samples_per_remote_writer** (p. 1622) or $\geq$ **DDS_RtpsReliableWriterProtocol_t::heartbeats_per_max_samples** (p. 1967)

For **DDS_DataWriterQos** (p. 1666) **max_samples** $\geq$ **DDS_RtpsReliableWriterProtocol_t::heartbeats_per_max_samples** (p. 1967) in **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 1653) if batching is disabled.

Categories

Immutable (p. 914), **Durability** (p. 860), **Sample Memory Management** (p. 902), **Middleware Memory Management** (p. 878), **Data Availability** (p. 843)

6.168.4.2 max_instances

`DDS_Long DDS_ResourceLimitsQosPolicy::max_instances`

<<**basic**>> (p. 783) <<**spec**>> (p. 790) Represents the maximum number of instances a **DDS_DataWriter** (p. 485) (or **DDS_DataReader** (p. 615)) can manage.

[default] **DDS_LENGTH_UNLIMITED** (p. 1326)

[range] [1, 1 million] or **DDS_LENGTH_UNLIMITED** (p. 1326), >= initial_instances

Categories

Immutable (p. 914), **Durability** (p. 860), **Instance Lifecycle** (p. 867), **Sample Memory Management** (p. 902), **Middleware Memory Management** (p. 878), **Data Availability** (p. 843)

6.168.4.3 max_samples_per_instance

`DDS_Long DDS_ResourceLimitsQosPolicy::max_samples_per_instance`

<<**spec**>> (p. 790) Represents the maximum number of samples of any one instance a **DDS_DataWriter** (p. 485) (or **DDS_DataReader** (p. 615)) 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. 1326).

[default] **DDS_LENGTH_UNLIMITED** (p. 1326)

[range] [1, 100 million] or **DDS_LENGTH_UNLIMITED** (p. 1326), <= max_samples or **DDS_LENGTH_UNLIMITED** (p. 1326), >= **DDS_HistoryQosPolicy::depth** (p. 1807)

Categories

Immutable (p. 914), **Durability** (p. 860), **Instance Lifecycle** (p. 867), **Sample Memory Management** (p. 902), **Middleware Memory Management** (p. 878), **Data Availability** (p. 843)

6.168.4.4 initial_samples

`DDS_Long DDS_ResourceLimitsQosPolicy::initial_samples`

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Represents the initial samples the middleware will store for any one **DDS_DataWriter** (p. 485) (or **DDS_DataReader** (p. 615)).

Specifies the initial number of data samples a **DDS_DataWriter** (p. 485) (or **DDS_DataReader** (p. 615)) will manage across all the instances associated with it.

[default] 32

[range] [1, 100 million], <= max_samples

Categories

Immutable (p. 914), **Sample Memory Management** (p. 902), **Middleware Memory Management** (p. 878)

6.168.4.5 initial_instances

`DDS_Long DDS_ResourceLimitsQosPolicy::initial_instances`

<<*basic*>> (p. 783) <<*extension*>> (p. 795) Represents the initial number of instances a **DDS_DataWriter** (p. 485) (or **DDS_DataReader** (p. 615)) will manage.

[default] 32

[range] [1,1 million], <= max_instances

Categories

Immutable (p. 914), **Instance Lifecycle** (p. 867), **Sample Memory Management** (p. 902), **Middleware Memory Management** (p. 878)

6.168.4.6 instance_hash_buckets

`DDS_Long DDS_ResourceLimitsQosPolicy::instance_hash_buckets`

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) 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. 914), **Instance Lifecycle** (p. 867)

6.169 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.

6.169.1 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.

6.170 DDS_RTPS_GUID_t Struct Reference

From the DDS-RTPS specification: type used to hold a globally-unique RTPS-entity identifier.

6.170.1 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.

6.171 DDS_RtpsReliableReaderProtocol_t Struct Reference

<<*extension*>> (p. 795) Qos related to reliable reader protocol defined in RTPS.

Data Fields

- struct **DDS_Duration_t min_heartbeat_response_delay**
 <<*extension*>> (p. 795) *The minimum delay to respond to a heartbeat.*
- struct **DDS_Duration_t max_heartbeat_response_delay**
 <<*extension*>> (p. 795) *The maximum delay to respond to a heartbeat.*
- struct **DDS_Duration_t heartbeat_suppression_duration**
 <<*extension*>> (p. 795) *The duration a reader ignores consecutively received heartbeats*
- struct **DDS_Duration_t nack_period**
 <<*extension*>> (p. 795) *The period at which to send NACKs*
- **DDS_Long receive_window_size**
 <<*extension*>> (p. 795) *The number of received out-of-order samples a reader can keep at a time*
- struct **DDS_Duration_t round_trip_time**
 <<*extension*>> (p. 795) *The duration from sending a NACK to receiving a repair of a sample*
- struct **DDS_Duration_t app_ack_period**
 <<*extension*>> (p. 795) *The period at which application-level acknowledgment messages are sent*
- struct **DDS_Duration_t min_app_ack_response_keep_duration**
 <<*extension*>> (p. 795) *Minimum duration for which application-level acknowledgment response data is kept*
- **DDS_Long samples_per_app_ack**
 <<*extension*>> (p. 795) *The minimum number of samples acknowledged by one application-level acknowledgment message*

6.171.1 Detailed Description

<<**extension**>> (p. 795) Qos related to reliable reader protocol defined in RTPS.

It is used to config reliable reader according to RTPS protocol.

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

QoS:

DDS_DataReaderProtocolQosPolicy (p. 1596) **DDS_DiscoveryConfigQosPolicy** (p. 1689)

6.171.2 Field Documentation

6.171.2.1 min_heartbeat_response_delay

```
struct DDS_Duration_t DDS_RtpsReliableReaderProtocol_t::min_heartbeat_response_delay
```

<<**extension**>> (p. 795) 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. 914), **Reliability** (p. 895)

6.171.2.2 max_heartbeat_response_delay

```
struct DDS_Duration_t DDS_RtpsReliableReaderProtocol_t::max_heartbeat_response_delay
```

<<**extension**>> (p. 795) 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. 1599): 0.5 seconds
- For **DDS_DiscoveryConfigQosPolicy::publication_reader** (p. 1696): 0 seconds
- For **DDS_DiscoveryConfigQosPolicy::subscription_reader** (p. 1697): 0 seconds
- For **DDS_DiscoveryConfigQosPolicy::participant_message_reader** (p. 1702): 0 seconds

[range] [0, 1 year], >= min_heartbeat_response_delay

Categories

Immutable (p. 914), **Reliability** (p. 895)

6.171.2.3 heartbeat_suppression_duration

```
struct DDS_Duration_t DDS_RtpsReliableReaderProtocol_t::heartbeat_suppression_duration
```

<<**extension**>> (p. 795) 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. 914), **Reliability** (p. 895)

6.171.2.4 nack_period

```
struct DDS_Duration_t DDS_RtpsReliableReaderProtocol_t::nack_period
```

<<**extension**>> (p. 795) 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. 914), **Reliability** (p. 895)

6.171.2.5 receive_window_size

```
DDS_Long DDS_RtpsReliableReaderProtocol_t::receive_window_size
```

<<**extension**>> (p. 795) 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. 914), **Reliability** (p. 895)

6.171.2.6 round_trip_time

```
struct DDS_Duration_t DDS_RtpsReliableReaderProtocol_t::round_trip_time
```

<<**extension**>> (p. 795) 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. 914), **Reliability** (p. 895)

6.171.2.7 app_ack_period

```
struct DDS_Duration_t DDS_RtpsReliableReaderProtocol_t::app_ack_period
```

<<**extension**>> (p. 795) The period at which application-level acknowledgment messages are sent

A **DDS_DataReader** (p. 615) sends application-level acknowledgment messages to a **DDS_DataWriter** (p. 485) at this periodic rate, and will continue sending until it receives a message from the **DDS_DataWriter** (p. 485) that it has received and processed the acknowledgment and an AppAckConfirmation has been received by the **DDS_DataReader** (p. 615). Note: application-level acknowledgment messages can also be sent non-periodically, as determined by **DDS_RtpsReliableReaderProtocol_t::samples_per_app_ack** (p. 1959).

[default] 5 seconds

[range] [1 nanosec, 1 year]

Categories

Immutable (p. 914), **Reliability** (p. 895), **Application Acknowledgement** (p. 835)

6.171.2.8 min_app_ack_response_keep_duration

```
struct DDS_Duration_t DDS_RtpsReliableReaderProtocol_t::min_app_ack_response_keep_duration
```

<<**extension**>> (p. 795) Minimum duration for which application-level acknowledgment response data is kept

The user-specified response data of an explicit application-level acknowledgment (called by **DDS_DataReader_↔_acknowledge_sample** (p. 677) or **DDS_DataReader_acknowledge_all** (p. 677)) is cached by the **DDS_DataReader** (p. 615) 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. 914), **Reliability** (p. 895), **Application Acknowledgement** (p. 835)

6.171.2.9 samples_per_app_ack

```
DDS_Long DDS_RtpsReliableReaderProtocol_t::samples_per_app_ack
```

<<**extension**>> (p. 795) The minimum number of samples acknowledged by one application-level acknowledgment message

This setting applies only when **DDS_ReliabilityQosPolicy::acknowledgment_kind** (p. 1939) = **DDS_APPLICATION_↔_EXPLICIT_ACKNOWLEDGMENT_MODE** (p. 1324) or **DDS_APPLICATION_AUTO_ACKNOWLEDGMENT_MODE** (p. 1323)

A **DDS_DataReader** (p. 615) 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 **DDS_DataReader_acknowledge_sample** (p. 677) this many times consecutively will trigger the sending of an acknowledgment message. Calling **DDS_DataReader_acknowledge_all** (p. 677) 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. 1958), where a **DDS_Data_↔_Reader** (p. 615) will send acknowledgement messages at the periodic rate regardless.

When this is set to **DDS_LENGTH_UNLIMITED** (p. 1326), then acknowledgement messages are sent only periodically, at the rate set by **DDS_RtpsReliableReaderProtocol_t::app_ack_period** (p. 1958).

[default] 1

[range] [1, 1000000], or **DDS_LENGTH_UNLIMITED** (p. 1326)

Categories

Immutable (p. 914), **Reliability** (p. 895), **Application Acknowledgement** (p. 835)

6.172 DDS_RtpsReliableWriterProtocol_t Struct Reference

<<*extension*>> (p. 795) QoS related to the reliable writer protocol defined in RTPS.

Data Fields

- **DDS_Long low_watermark**
 <<*extension*>> (p. 795) 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. 1234) is considered to have changed.
- **DDS_Long high_watermark**
 <<*extension*>> (p. 795) 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. 1234) is considered to have changed.
- struct **DDS_Duration_t heartbeat_period**
 <<*basic*>> (p. 783) <<*extension*>> (p. 795) The period at which to send heartbeats.
- struct **DDS_Duration_t fast_heartbeat_period**
 <<*basic*>> (p. 783) <<*extension*>> (p. 795) 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. 783) <<*extension*>> (p. 795) 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. 795) 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. 795) The number of samples that a reliable writer has to publish before sending a virtual heartbeat.
- **DDS_Long max_heartbeat_retries**
 <<*extension*>> (p. 795) The maximum number of periodic heartbeat retries before marking a remote reader as inactive.
- **DDS_Boolean inactivate_nonprogressing_readers**
 <<*extension*>> (p. 795) Whether to treat remote readers as inactive when their NACKs do not progress
- **DDS_Long heartbeats_per_max_samples**
 <<*extension*>> (p. 795) The number of piggyback heartbeats sent per max send window.
- struct **DDS_Duration_t min_nack_response_delay**
 <<*extension*>> (p. 795) The minimum delay to respond to a NACK or NACK_FRAG.
- struct **DDS_Duration_t max_nack_response_delay**
 <<*extension*>> (p. 795) The maximum delay to respond to a NACK or NACK_FRAG.
- struct **DDS_Duration_t nack_suppression_duration**
 <<*extension*>> (p. 795) The duration for ignoring consecutive NACKs/NACK_FRAGs that may trigger redundant re-pairs.
- **DDS_Long max_bytes_per_nack_response**
 <<*extension*>> (p. 795) 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. 795) The minimum duration a sample is queued for ACK-disabled readers.
- struct **DDS_Duration_t disable_positive_acks_max_sample_keep_duration**
 <<*extension*>> (p. 795) The maximum duration a sample is queued for ACK-disabled readers.
- **DDS_Boolean disable_positive_acks_enable_adaptive_sample_keep_duration**

[DEPRECATED] (p. 965) <<extension>> (p. 795) Enables dynamic adjustment of sample keep duration in response to congestion.

- **DDS_Long disable_positive_acks_decrease_sample_keep_duration_factor**
[DEPRECATED] (p. 965) <<extension>> (p. 795) Controls rate of contraction of dynamic sample keep duration.
- **DDS_Long disable_positive_acks_increase_sample_keep_duration_factor**
[DEPRECATED] (p. 965) <<extension>> (p. 795) Controls rate of growth of dynamic sample keep duration.
- **DDS_Long min_send_window_size**
 <<basic>> (p. 783) <<extension>> (p. 795) Minimum size of send window of unacknowledged samples
- **DDS_Long max_send_window_size**
 <<basic>> (p. 783) <<extension>> (p. 795) Maximum size of send window of unacknowledged samples
- struct **DDS_Duration_t send_window_update_period**
 <<extension>> (p. 795) Period in which send window may be dynamically changed
- **DDS_Long send_window_increase_factor**
 <<extension>> (p. 795) Increases send window size by this percentage when reacting dynamically to network conditions
- **DDS_Long send_window_decrease_factor**
 <<extension>> (p. 795) Decreases send window size by this percentage when reacting dynamically to network conditions
- **DDS_Boolean enable_multicast_periodic_heartbeat**
 <<extension>> (p. 795) Whether periodic heartbeat messages are sent over multicast
- **DDS_Long multicast_resend_threshold**
 <<extension>> (p. 795) The minimum number of requesting readers needed to trigger a multicast resend
- **DDS_Boolean disable_repair_piggyback_heartbeat**
 <<extension>> (p. 795) Prevents piggyback heartbeats from being sent with repair samples

6.172.1 Detailed Description

<<extension>> (p. 795) 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. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

QoS:

DDS_DataWriterProtocolQosPolicy (p. 1649) **DDS_DiscoveryConfigQosPolicy** (p. 1689)

6.172.2 Field Documentation

6.172.2.1 low_watermark

DDS_Long DDS_RtpsReliableWriterProtocol_t::low_watermark

<<**extension**>> (p. 795) 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. 1234) 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_RtpsReliableWriterProtocol_t::heartbeat_period** (p. 1963) and **DDS_RtpsReliableWriterProtocol_t::fast_heartbeat_period** (p. 1963), respectively). When the number of unacknowledged samples in the queue of a reliable **DDS_DataWriter** (p. 485) meets or exceeds high_watermark, the **DDS_RELIABLE_WRITER_CACHE_CHANGED_STATUS** (p. 1234) is changed, and the DataWriter will start heartbeating at **DDS_RtpsReliableWriterProtocol_t::fast_heartbeat_period** (p. 1963). When the number of samples meets or falls below low_watermark, **DDS_RELIABLE_WRITER_CACHE_CHANGED_STATUS** (p. 1234) is changed, and the heartbeat rate will return to the "normal" rate (**DDS_RtpsReliableWriterProtocol_t::heartbeat_period** (p. 1963)).

[default] 0

[range] [0, 100 million], < high_watermark

Categories

Immutable (p. 914), **Reliability** (p. 895)

6.172.2.2 high_watermark

DDS_Long DDS_RtpsReliableWriterProtocol_t::high_watermark

<<**extension**>> (p. 795) 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. 1234) 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. 1730):

For **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 1698)
 high_watermark ≤ **DDS_AllocationSettings_t::max_count** (p. 1538) in **DDS_DomainParticipantResourceLimitsQosPolicy::local_writer_allocation** (p. 1735)

For **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 1700)
 high_watermark <= **DDS_AllocationSettings_t::max_count** (p. 1538) in **DDS_DomainParticipantResourceLimits** ↔
QosPolicy::local_reader_allocation (p. 1735)

In **DDS_DataWriterQos::protocol** (p. 1671):

For **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 1653),

high_watermark <= **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1951) if batching is disabled. Otherwise,
 high_watermark <= **DDS_DataWriterResourceLimitsQosPolicy::max_batches** (p. 1677) high_watermark <= **DDS_** ↔
RtpsReliableWriterProtocol_t::max_send_window_size (p. 1973)

[default] 1

[range] [1, 100 million] or **DDS_LENGTH_UNLIMITED** (p. 1326), > low_watermark <= *maximum* which depends on the container policy

Categories

Immutable (p. 914), **Reliability** (p. 895)

6.172.2.3 heartbeat_period

```
struct DDS_Duration_t DDS_RtpsReliableWriterProtocol_t::heartbeat_period
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) 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. 1653): 200 ms
- For **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 1698): 3.0 seconds
- For **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 1700): 3.0 seconds
- For **DDS_DiscoveryConfigQosPolicy::participant_message_writer** (p. 1702): 1.0 seconds

[range] [1 nanosec, 1 year], >= **DDS_RtpsReliableWriterProtocol_t::fast_heartbeat_period** (p. 1963), >= **DDS_** ↔
RtpsReliableWriterProtocol_t::late_joiner_heartbeat_period (p. 1964)

Categories

Immutable (p. 914), **Reliability** (p. 895)

6.172.2.4 fast_heartbeat_period

```
struct DDS_Duration_t DDS_RtpsReliableWriterProtocol_t::fast_heartbeat_period
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) 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. 1653): 200 ms seconds
- For **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 1698): 3.0 seconds
- For **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 1700): 3.0 seconds
- For **DDS_DiscoveryConfigQosPolicy::participant_message_writer** (p. 1702): 1.0 seconds

[range] [1 nanosec, 1 year], <= **DDS_RtpsReliableWriterProtocol_t::heartbeat_period** (p. 1963)

Categories

Immutable (p. 914), **Reliability** (p. 895)

6.172.2.5 late_joiner_heartbeat_period

```
struct DDS_Duration_t DDS_RtpsReliableWriterProtocol_t::late_joiner_heartbeat_period
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) 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. 1964), to service the late joining reader. This means that if the **DDS_RtpsReliableWriterProtocol_t::fast_heartbeat_period** (p. 1963) 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. 1653): 200 seconds
- For **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 1698): 3.0 seconds
- For **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 1700): 3.0 seconds
- For **DDS_DiscoveryConfigQosPolicy::participant_message_writer** (p. 1702): 1.0 seconds

[range] [1 nanosec, 1 year], <= **DDS_RtpsReliableWriterProtocol_t::heartbeat_period** (p. 1963)

Categories

Immutable (p. 914), **Reliability** (p. 895)

6.172.2.6 virtual_heartbeat_period

```
struct DDS_Duration_t DDS_RtpsReliableWriterProtocol_t::virtual_heartbeat_period
```

<<**extension**>> (p. 795) 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. 1653): **DDS_DURATION_AUTO** (p. 1209). If **DDS_PresentationQosPolicy::access_scope** (p. 1893) is set to **DDS_GROUP_PRESENTATION_QOS** (p. 1305) on the DataWriter, this value is set to **DDS_RtpsReliableWriterProtocol_t::heartbeat_period** (p. 1963). Otherwise, the value is set to **DDS_DURATION_INFINITE** (p. 1208).
- For **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 1698): **DDS_DURATION_INFINITE** (p. 1208)
- For **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 1700): **DDS_DURATION_INFINITE** (p. 1208)
- For **DDS_DiscoveryConfigQosPolicy::participant_message_writer** (p. 1702): **DDS_DURATION_INFINITE** (p. 1208)

[range] > 1 nanosec, **DDS_DURATION_INFINITE** (p. 1208), or **DDS_DURATION_AUTO** (p. 1209)

Categories

Immutable (p. 914), **Reliability** (p. 895)

6.172.2.7 samples_per_virtual_heartbeat

DDS_Long DDS_RtpsReliableWriterProtocol_t::samples_per_virtual_heartbeat

<<**extension**>> (p. 795) The number of samples that a reliable writer has to publish before sending a virtual heartbeat.

[default] DDS_LENGTH_UNLIMITED (p. 1326)

[range] [1,1000000], DDS_LENGTH_UNLIMITED (p. 1326)

Categories

Immutable (p. 914), **Reliability** (p. 895)

6.172.2.8 max_heartbeat_retries

DDS_Long DDS_RtpsReliableWriterProtocol_t::max_heartbeat_retries

<<**extension**>> (p. 795) 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. 1326)

Categories

Immutable (p. 914), **Reliability** (p. 895), **Liveliness** (p. 874)

6.172.2.9 `inactivate_nonprogressing_readers`

DDS_Boolean DDS_RtpsReliableWriterProtocol_t::inactivate_nonprogressing_readers

<<**extension**>> (p. 795) 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. 1966) have been sent without receiving any ack/nacks back.

By setting this **DDS_BOOLEAN_TRUE** (p. 1201), 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. 1966) 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. 1201)

Categories

Immutable (p. 914), **Reliability** (p. 895), **Liveliness** (p. 874)

6.172.2.10 `heartbeats_per_max_samples`

DDS_Long DDS_RtpsReliableWriterProtocol_t::heartbeats_per_max_samples

<<**extension**>> (p. 795) 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. 1973) 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. 1967) 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. 1951), **DDS_RtpsReliableWriterProtocol_t::max_send_window_size** (p. 1973))

With batching:

min (**DDS_DataWriterResourceLimitsQosPolicy::max_batches** (p. 1677), **DDS_RtpsReliableWriterProtocol_t::max_send_window_size** (p. 1973))

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. 1326), 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. 1653): 8
- For **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 1698): 8
- For **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 1700): 8
- For **DDS_DiscoveryConfigQosPolicy::participant_message_writer** (p. 1702): 1

[range] [0, 100 million]

- For **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 1698):
heartbeats_per_max_samples ≤ **DDS_AllocationSettings_t::max_count** (p. 1538) in **DDS_Domain** ↔ **ParticipantResourceLimitsQosPolicy::local_writer_allocation** (p. 1735)
- For **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 1700):
heartbeats_per_max_samples ≤ **DDS_AllocationSettings_t::max_count** (p. 1538) in **DDS_Domain** ↔ **ParticipantResourceLimitsQosPolicy::local_reader_allocation** (p. 1735)
- For **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 1653):

heartbeats_per_max_samples ≤ **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1951) if batching is disabled. Otherwise:

heartbeats_per_max_samples ≤ **DDS_DataWriterResourceLimitsQosPolicy::max_batches** (p. 1677)

heartbeats_per_max_samples ≤ **DDS_RtpsReliableWriterProtocol_t::max_send_window_size** (p. 1973)

Categories

Immutable (p. 914), **Reliability** (p. 895)

6.172.2.11 min_nack_response_delay

```
struct DDS_Duration_t DDS_RtpsReliableWriterProtocol_t::min_nack_response_delay
```

<<**extension**>> (p. 795) 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. 914), **Reliability** (p. 895), **Fragmentation** (p. 866)

6.172.2.12 max_nack_response_delay

```
struct DDS_Duration_t DDS_RtpsReliableWriterProtocol_t::max_nack_response_delay
```

<<**extension**>> (p. 795) 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. 1653): 0.2 seconds
- For **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 1698): 0 seconds
- For **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 1700): 0 seconds
- For **DDS_DiscoveryConfigQosPolicy::participant_message_writer** (p. 1702): 0 seconds

[range] [0,1 day], >= min_nack_response_delay

Categories

Immutable (p. 914), **Reliability** (p. 895), **Fragmentation** (p. 866)

6.172.2.13 nack_suppression_duration

```
struct DDS_Duration_t DDS_RtpsReliableWriterProtocol_t::nack_suppression_duration
```

<<**extension**>> (p. 795) 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. 914), **Reliability** (p. 895), **Fragmentation** (p. 866)

6.172.2.14 max_bytes_per_nack_response

DDS_Long DDS_RtpsReliableWriterProtocol_t::max_bytes_per_nack_response

<<**extension**>> (p. 795) 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. 1968) and **DDS_RtpsReliableWriterProtocol_t::max_nack_response_delay** (p. 1968) are both **DDS_DURATION_ZERO** (p. 1209) (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. 1653): 131072 bytes
- For **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 1698): 131072 bytes
- For **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 1700): 131072 bytes
- For **DDS_DiscoveryConfigQosPolicy::participant_message_writer** (p. 1702): 9216 bytes

[range] [0, 1 GB]

Categories

Immutable (p. 914), **Reliability** (p. 895), **Flow Control** (p. 863), **Fragmentation** (p. 866)

6.172.2.15 disable_positive_acks_min_sample_keep_duration

```
struct DDS_Duration_t DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_min_sample_keep_↵  
duration
```

<<**extension**>> (p. 795) 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. 1651) = **DDS_BOOLEAN_TRUE** (p. 1201)) or a data reader (**DDS_DataReaderProtocolQosPolicy::disable_↵
positive_acks** (p. 1598) = **DDS_BOOLEAN_TRUE** (p. 1201)), 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. 1971)

Categories

Immutable (p. 914), **Reliability** (p. 895), **Durability** (p. 860)

6.172.2.16 disable_positive_acks_max_sample_keep_duration

```
struct DDS_Duration_t DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_max_sample_keep_↵  
duration
```

<<**extension**>> (p. 795) 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. 1651) = **DDS_BOOLEAN_TRUE** (p. 1201)) or a data reader (**DDS_DataReaderProtocolQosPolicy::disable_↵
positive_acks** (p. 1598) = **DDS_BOOLEAN_TRUE** (p. 1201)), 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. 1970)

Categories

Immutable (p. 914), **Reliability** (p. 895), **Durability** (p. 860)

6.172.2.17 disable_positive_acks_enable_adaptive_sample_keep_duration

DDS_Boolean DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_enable_adaptive_sample_keep_duration

[DEPRECATED] (p. 965) <<extension>> (p. 795) 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. 1971) = **DDS_BOOLEAN_TRUE** (p. 1201), 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. 1972). 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. 1972).

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. 1970) and **DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_max_sample_keep_duration** (p. 1971), respectively.

When **DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_enable_adaptive_sample_keep_duration** (p. 1971) = **DDS_BOOLEAN_FALSE** (p. 1201), the sample keep duration is set to **DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_min_sample_keep_duration** (p. 1970).

[default] **DDS_BOOLEAN_TRUE** (p. 1201)

Categories

Immutable (p. 914), **Reliability** (p. 895), **Durability** (p. 860)

6.172.2.18 disable_positive_acks_decrease_sample_keep_duration_factor

DDS_Long DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_decrease_sample_keep_duration_factor

[DEPRECATED] (p. 965) <<extension>> (p. 795) Controls rate of contraction of dynamic sample keep duration.

Used when **DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_enable_adaptive_sample_keep_duration** (p. 1971) = **DDS_BOOLEAN_TRUE** (p. 1201).

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. 914), **Reliability** (p. 895), **Durability** (p. 860)

6.172.2.19 disable_positive_acks_increase_sample_keep_duration_factor

DDS_Long DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_increase_sample_keep_duration_factor

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Controls rate of growth of dynamic sample keep duration.

Used when **DDS_RtpsReliableWriterProtocol_t::disable_positive_acks_enable_adaptive_sample_keep_duration** (p. 1971) = **DDS_BOOLEAN_TRUE** (p. 1201).

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. 914), **Reliability** (p. 895), **Durability** (p. 860)

6.172.2.20 min_send_window_size

DDS_Long DDS_RtpsReliableWriterProtocol_t::min_send_window_size

<<*basic*>> (p. 783) <<*extension*>> (p. 795) Minimum size of send window of unacknowledged samples

A **DDS_DataWriter** (p. 485) 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. 1326)

[range] > 0, <= **max_send_window_size**, or **DDS_LENGTH_UNLIMITED** (p. 1326)

See also

DDS_RtpsReliableWriterProtocol_t::max_send_window_size (p. 1973)

DDS_RtpsReliableWriterProtocol_t::low_watermark (p. 1961)

DDS_RtpsReliableWriterProtocol_t::high_watermark (p. 1962)

DDS_ReliableWriterCacheChangedStatus::full_reliable_writer_cache (p. 1944)

Categories

Immutable (p. 914), **Reliability** (p. 895), **Flow Control** (p. 863)

6.172.2.21 max_send_window_size

DDS_Long DDS_RtpsReliableWriterProtocol_t::max_send_window_size

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Maximum size of send window of unacknowledged samples

A **DDS_DataWriter** (p. 485) 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. 1326), then either **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1951) (for non-batching) or **DDS_DataWriterResourceLimitsQosPolicy::max_batches** (p. 1677) (for batching) serves as the effective max_send_window_size. When **DDS_ResourceLimitsQosPolicy::max_samples** (p. 1951) (for non-batching) or **DDS_DataWriterResourceLimitsQosPolicy::max_batches** (p. 1677) (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. 1326)

[range] > 0, >= min_send_window_size, or **DDS_LENGTH_UNLIMITED** (p. 1326)

See also

DDS_RtpsReliableWriterProtocol_t::min_send_window_size (p. 1973)

DDS_RtpsReliableWriterProtocol_t::low_watermark (p. 1961)

DDS_RtpsReliableWriterProtocol_t::high_watermark (p. 1962)

DDS_ReliableWriterCacheChangedStatus::full_reliable_writer_cache (p. 1944)

Categories

Immutable (p. 914), **Reliability** (p. 895), **Flow Control** (p. 863)

6.172.2.22 send_window_update_period

```
struct DDS_Duration_t DDS_RtpsReliableWriterProtocol_t::send_window_update_period
```

<<**extension**>> (p. 795) Period in which send window may be dynamically changed

The **DDS_DataWriter** (p. 485)'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. 1653): 3 seconds
- For **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 1698): 3 seconds
- For **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 1700): 3 seconds
- For **DDS_DiscoveryConfigQosPolicy::participant_message_writer** (p. 1702): 1 second

[range] > [0,1 year]

See also

DDS_RtpsReliableWriterProtocol_t::send_window_increase_factor (p. 1975), **DDS_RtpsReliableWriterProtocol_t::send_window_decrease_factor** (p. 1976)

Categories

Immutable (p. 914), **Reliability** (p. 895), **Flow Control** (p. 863)

6.172.2.23 send_window_increase_factor

DDS_Long DDS_RtpsReliableWriterProtocol_t::send_window_increase_factor

<<**extension**>> (p. 795) Increases send window size by this percentage when reacting dynamically to network conditions

The **DDS_DataWriter** (p. 485)'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. 1974), **DDS_RtpsReliableWriterProtocol_t::send_window_decrease_factor** (p. 1976)

Categories

Immutable (p. 914), **Reliability** (p. 895), **Flow Control** (p. 863)

6.172.2.24 send_window_decrease_factor

DDS_Long DDS_RtpsReliableWriterProtocol_t::send_window_decrease_factor

<<**extension**>> (p. 795) Decreases send window size by this percentage when reacting dynamically to network conditions

The **DDS_DataWriter** (p. 485)'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. 1653): 70
- For **DDS_DiscoveryConfigQosPolicy::publication_writer** (p. 1698): 50
- For **DDS_DiscoveryConfigQosPolicy::subscription_writer** (p. 1700): 50
- For **DDS_DiscoveryConfigQosPolicy::participant_message_writer** (p. 1702): 50

[range] [0, 100]

See also

DDS_RtpsReliableWriterProtocol_t::send_window_update_period (p. 1974), **DDS_RtpsReliableWriterProtocol_t::send_window_increase_factor** (p. 1975)

Categories

Immutable (p. 914), **Reliability** (p. 895), **Flow Control** (p. 863)

6.172.2.25 enable_multicast_periodic_heartbeat

DDS_Boolean **DDS_RtpsReliableWriterProtocol_t::enable_multicast_periodic_heartbeat**

<<*extension*>> (p. 795) 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. 1201)

Categories

Immutable (p. 914), **Reliability** (p. 895), **Transports** (p. 908)

6.172.2.26 multicast_resend_threshold

DDS_Long DDS_RtpsReliableWriterProtocol_t::multicast_resend_threshold

<<**extension**>> (p. 795) 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] [$\geq$ 1]

Categories

Immutable (p. 914), **Reliability** (p. 895), **Transports** (p. 908)

6.172.2.27 disable_repair_piggyback_heartbeat

DDS_Boolean DDS_RtpsReliableWriterProtocol_t::disable_repair_piggyback_heartbeat

<<**extension**>> (p. 795) Prevents piggyback heartbeats from being sent with repair samples

When samples are repaired, the **DDS_DataWriter** (p. 485) resends **DDS_RtpsReliableWriterProtocol_t::max_bytes_per_nack_response** (p. 1969) bytes and a piggyback heartbeat with each message. You can configure the **DDS_DataWriter** (p. 485) to not send the piggyback heartbeat and instead rely on the **DDS_RtpsReliableWriterProtocol_t::late_joiner_heartbeat_period** (p. 1964) to control the throughput used to repair samples. This field is mutable only for **DDS_DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 1653). **[default]** **DDS_BOOLEAN_FALSE** (p. 1201)

Categories

Immutable (p. 914), **Reliability** (p. 895), **Flow Control** (p. 863)

6.173 DDS_RtpsWellKnownPorts_t Struct Reference

<<**extension**>> (p. 795) RTPS well-known port mapping configuration.

Data Fields

- **DDS_Long port_base**
 <<extension>> (p. 795) The base port offset.
- **DDS_Long domain_id_gain**
 <<extension>> (p. 795) Tunable domain gain parameter.
- **DDS_Long participant_id_gain**
 <<extension>> (p. 795) Tunable participant gain parameter.
- **DDS_Long builtin_multicast_port_offset**
 <<extension>> (p. 795) Additional offset for **metatraffic** multicast port.
- **DDS_Long builtin_unicast_port_offset**
 <<extension>> (p. 795) Additional offset for **metatraffic** unicast port.
- **DDS_Long user_multicast_port_offset**
 <<extension>> (p. 795) Additional offset for **usertraffic** multicast port.
- **DDS_Long user_unicast_port_offset**
 <<extension>> (p. 795) Additional offset for **usertraffic** unicast port.

6.173.1 Detailed Description

<<extension>> (p. 795) 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_RtpsWellKnownPorts_t** (p. 1978). 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. 1978), the port numbers depend on:

- `domain_id`, as specified in **DDS_DomainParticipantFactory_create_participant** (p. 41)
- `participant_id`, as specified using **DDS_WireProtocolQosPolicy::participant_id** (p. 2097)

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.

$$\text{metatraffic\_unicast\_port} = \text{port\_base} + (\text{domain\_id\_gain} * \text{domain\_id}) + (\text{participant\_id\_gain} * \text{participant\_id}) + \text{builtin\_unicast\_port\_offset}$$

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. 1717) on a **DDS_DomainParticipant** (p. 77) 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. 1616), **DDS_DataReaderQos::unicast** (p. 1616), or by the **DDS_DataWriterQos::unicast** (p. 1671) 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. 2058).

```
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. 1349)).

The OMG's DDS Interoperability Wire Protocol compliant port mapping parameters are *not* backwards compatible with previous versions of the RTI Connex 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 Connex'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](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. 2093)

6.173.2 Field Documentation

6.173.2.1 port_base

`DDS_Long DDS_RtpsWellKnownPorts_t::port_base`

<<**extension**>> (p. 795) The base port offset.

All mapped well-known ports are offset by this value.

[default] 7400

[range] [$\geq$ 1], but resulting ports must be within the range imposed by the underlying transport.

6.173.2.2 domain_id_gain

`DDS_Long DDS_RtpsWellKnownPorts_t::domain_id_gain`

<<**extension**>> (p. 795) 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 **DDS_DomainParticipant** (p. 77) instances within a single domain occupy a consecutive range of `domain_id_gain` ports. Precisely, all ports occupied by the domain fall within:

$$(\text{port\_base} + (\text{domain\_id\_gain} * \text{domain\_id}))$$

and:

$$(\text{port\_base} + (\text{domain\_id\_gain} * (\text{domain\_id} + 1)) - 1)$$

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:

$$\text{max\_participant\_id} < (\text{domain\_id\_gain} / \text{participant\_id\_gain})$$

On the contrary, if `domain_id_gain ≤ participant_id_gain`, it results in a port mapping layout where a given domain's **DDS_DomainParticipant** (p. 77) 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.

6.173.2.3 participant_id_gain

```
DDS_Long DDS_RtpsWellKnownPorts_t::participant_id_gain
```

<<*extension*>> (p. 795) Tunable participant gain parameter.

Multiplier of the `participant_id`. See **DDS_RtpsWellKnownPorts_t::domain_id_gain** (p. 1981) 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.

6.173.2.4 builtin_multicast_port_offset

```
DDS_Long DDS_RtpsWellKnownPorts_t::builtin_multicast_port_offset
```

<<*extension*>> (p. 795) 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.

6.173.2.5 builtin_unicast_port_offset

`DDS_Long DDS_RtpsWellKnownPorts_t::builtin_unicast_port_offset`

<<**extension**>> (p. 795) 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.

6.173.2.6 user_multicast_port_offset

`DDS_Long DDS_RtpsWellKnownPorts_t::user_multicast_port_offset`

<<**extension**>> (p. 795) 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.

6.173.2.7 user_unicast_port_offset

`DDS_Long DDS_RtpsWellKnownPorts_t::user_unicast_port_offset`

<<**extension**>> (p. 795) 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.

6.174 DDS_SampleHandler Struct Reference

<<**experimental**>> (p. 1012) <<**extension**>> (p. 1012) <<**interface**>> (p. 1013) Handler called by a sample dispatcher, such as **DDS_SampleProcessor** (p. 1505).

Data Fields

- void * **handler_data**

A place for handler implementors to keep a pointer to data that may be needed by their handler.

- DDS_SampleHandler_OnNewSampleCallback **on_new_sample**

*Handles the dispatch of a sample read from the associated **DDS_DataReader** (p. 615).*

6.174.1 Detailed Description

<<*experimental*>> (p. 1012) <<*extension*>> (p. 1012) <<*interface*>> (p. 1013) Handler called by a sample dispatcher, such as **DDS_SampleProcessor** (p. 1505).

6.174.2 Field Documentation

6.174.2.1 handler_data

```
void* DDS_SampleHandler::handler_data
```

A place for handler implementors to keep a pointer to data that may be needed by their handler.

6.174.2.2 on_new_sample

```
DDS_SampleHandler_OnNewSampleCallback DDS_SampleHandler::on_new_sample
```

Handles the dispatch of a sample read from the associated **DDS_DataReader** (p. 615).

This operation is called from the dispatcher after reading a new sample from an associated **DDS_DataReader** (p. 615). Note that the type of the data portion of the Sample must correspond with the type of the associated **DDS_DataReader** (p. 615).

6.175 DDS_SampleIdentity_t Struct Reference

Type definition for a Sample Identity.

Data Fields

- 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.

6.175.1 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.

6.176 DDS_SampleInfo Struct Reference

Information that accompanies each sample that is `read` or `taken`.

Data Fields

- **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's state of the instance.
- struct **DDS_Time_t reception_timestamp**
<<extension>> (p. 1012) The timestamp when the sample was committed by a DataReader.
- struct **DDS_SequenceNumber_t publication_sequence_number**
<<extension>> (p. 1012) The publication sequence number.
- struct **DDS_SequenceNumber_t reception_sequence_number**
<<extension>> (p. 1012) The reception sequence number when sample was committed by a DataReader
- struct **DDS_GUID_t original_publication_virtual_guid**
<<extension>> (p. 1012) The original publication virtual GUID.
- struct **DDS_SequenceNumber_t original_publication_virtual_sequence_number**
<<extension>> (p. 1012) The original publication virtual sequence number.
- struct **DDS_GUID_t related_original_publication_virtual_guid**
<<extension>> (p. 1012) The original publication virtual GUID of a related sample.
- struct **DDS_SequenceNumber_t related_original_publication_virtual_sequence_number**
<<extension>> (p. 1012) The original publication virtual sequence number of a related sample.
- **DDS_SampleFlag flag**

- `<<extension>>` (p. 1012) *Flags associated with the sample.*
- struct **DDS_GUID_t source_guid**
 - `<<extension>>` (p. 1012) *The application logical data source associated with the sample.*
- struct **DDS_GUID_t related_source_guid**
 - `<<extension>>` (p. 1012) *The application logical data source that is related to the sample.*
- struct **DDS_GUID_t related_subscription_guid**
 - `<<extension>>` (p. 1012) *The related_reader_guid associated with the sample.*
- struct **DDS_GUID_t topic_query_guid**
 - `<<extension>>` (p. 1012) *The GUID of the **DDS_TopicQuery** (p. 707) that is related to the sample.*
- struct **DDS_CoherentSetInfo_t * coherent_set_info**
 - `<<extension>>` (p. 1012) *The information about the coherent set that this sample is a part of.*

6.176.1 Detailed Description

Information that accompanies each sample that is `read` or `taken`.

6.176.2 Interpretation of the SampleInfo

The **DDS_SampleInfo** (p. 1985) 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. 1992) 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. 715)) 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 **DDS_DataReader** (p. 615). This rank provides a preview of what is available within the **DDS_DataReader** (p. 615).
- The `source_timestamp` of the sample. This is the timestamp provided by the **DDS_DataWriter** (p. 485) at the time the sample was produced.

6.176.3 Interpretation of the SampleInfo disposed_generation_count and no_writers_generation_count

For each instance, RTI Connexx internally maintains two counts, the **DDS_SampleInfo::disposed_generation_count** (p. 1990) and **DDS_SampleInfo::no_writers_generation_count** (p. 1990), relative to each DataReader:

- The **DDS_SampleInfo::disposed_generation_count** (p. 1990) and **DDS_SampleInfo::no_writers_generation_count** (p. 1990) are initialized to zero when the **DDS_DataReader** (p. 615) first detects the presence of a never-seen-before instance.
- The **DDS_SampleInfo::disposed_generation_count** (p. 1990) is incremented each time the instance_state of the corresponding instance changes from **DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 715) to **DDS_ALIVE_INSTANCE_STATE** (p. 715).
- The **DDS_SampleInfo::no_writers_generation_count** (p. 1990) is incremented each time the instance_state of the corresponding instance changes from **DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 715) to **DDS_ALIVE_INSTANCE_STATE** (p. 715).
- These 'generation counts' are reset to zero when the instance resource is reclaimed.

The **DDS_SampleInfo::disposed_generation_count** (p. 1990) and **DDS_SampleInfo::no_writers_generation_count** (p. 1990) available in the **DDS_SampleInfo** (p. 1985) capture a snapshot of the corresponding counters at the time the sample was received.

6.176.4 Interpretation of the SampleInfo sample_rank, generation_rank and absolute_generation_rank

The **DDS_SampleInfo::sample_rank** (p. 1991) and **DDS_SampleInfo::generation_rank** (p. 1991) available in the **DDS_SampleInfo** (p. 1985) are computed based solely on the actual samples in the ordered collection returned by read or take.

- The **DDS_SampleInfo::sample_rank** (p. 1991) indicates the number of samples of the same instance that follow the current one in the collection.
- The **DDS_SampleInfo::generation_rank** (p. 1991) available in the **DDS_SampleInfo** (p. 1985) 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. 1991) 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. 1991) available in the **DDS_SampleInfo** (p. 1985) indicates the difference in "generations" between the sample (S) and the Most Recent Sample of the same instance that the middle-ware 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)

```

6.176.5 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. 1985), 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. 710), **DDS_InstanceStateKind** (p. 714), **DDS_ViewStateKind** (p. 712), **DDS_SampleInfo::valid_data** (p. 1992)

"Statechart of the \p instance_state and \p view_state of a single instance"

6.176.6 Field Documentation

6.176.6.1 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. 710)

6.176.6.2 view_state

DDS_ViewStateKind DDS_SampleInfo::view_state

The view state of the instance.

Indicates whether the **DDS_DataReader** (p. 615) has already seen samples for the most-current generation of the related instance.

See also

DDS_ViewStateKind (p. 712)

6.176.6.3 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. 714)

6.176.6.4 source_timestamp

```
struct DDS_Time_t DDS_SampleInfo::source_timestamp
```

The timestamp when the sample was written by a DataWriter.

6.176.6.5 instance_handle

```
DDS_InstanceHandle_t DDS_SampleInfo::instance_handle
```

Identifies locally the corresponding instance.

The handle is equal to **DDS_HANDLE_NIL** (p. 228) for unkeyed topics.

6.176.6.6 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. 214) that is returned by the operation **DDS_DataReader_get_matched_publications** (p. 678) and can also be used as a parameter to the operation **DDS_DataReader_get_matched_publication_data** (p. 679).

6.176.6.7 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. 715) to **DDS_ALIVE_INSTANCE_STATE** (p. 715). 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. 1987)
Interpretation of the SampleInfo counters and ranks (p. 1988)

6.176.6.8 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. 715) to **DDS_ALIVE_INSTANCE_STATE** (p. 715). 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. 1987)
Interpretation of the SampleInfo counters and ranks (p. 1988)

6.176.6.9 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. 1987)
Interpretation of the SampleInfo counters and ranks (p. 1988)

6.176.6.10 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. 1987)
Interpretation of the SampleInfo counters and ranks (p. 1988)

6.176.6.11 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. 1987)
Interpretation of the SampleInfo counters and ranks (p. 1988)

6.176.6.12 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. 1985) and some Data. However there are situations where a `DataSample` contains only the **DDS_SampleInfo** (p. 1985) and does not have any associated data. This occurs when the RTI Connexx 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 Connexx detects that an instance has no writers and changes the corresponding `instance_state` to **DDS_NOT_↵ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 715).

The application can distinguish whether a particular `DataSample` has data by examining the value of the **DDS_↵SampleInfo::valid_data** (p. 1992). If this flag is set to **DDS_BOOLEAN_TRUE** (p. 1201), then the `Data_↵Sample` contains valid Data. If the flag is set to **DDS_BOOLEAN_FALSE** (p. 1201), 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. 1201), the application should not access the Data associated with the `DataSample`, that is, the application should access only the **DDS_↵SampleInfo** (p. 1985).

6.176.6.13 reception_timestamp

struct DDS_Time_t DDS_SampleInfo::reception_timestamp

<<*extension*>> (p. 1012) The timestamp when the sample was committed by a `DataReader`.

6.176.6.14 publication_sequence_number

```
struct DDS_SequenceNumber_t DDS_SampleInfo::publication_sequence_number
```

<<*extension*>> (p. 1012) The publication sequence number.

6.176.6.15 reception_sequence_number

```
struct DDS_SequenceNumber_t DDS_SampleInfo::reception_sequence_number
```

<<*extension*>> (p. 1012) The reception sequence number when sample was committed by a DataReader

6.176.6.16 original_publication_virtual_guid

```
struct DDS_GUID_t DDS_SampleInfo::original_publication_virtual_guid
```

<<*extension*>> (p. 1012) The original publication virtual GUID.

If the **DDS_PresentationQosPolicy::access_scope** (p. 1893) of the **DDS_Publisher** (p. 443) is **DDS_GROUP_PPRESENTATION_QOS** (p. 1305), this field contains the **DDS_Publisher** (p. 443) virtual GUID that uniquely identifies the DataWriter group.

See also

DDS_SampleInfo_get_sample_identity (p. 702)

6.176.6.17 original_publication_virtual_sequence_number

```
struct DDS_SequenceNumber_t DDS_SampleInfo::original_publication_virtual_sequence_number
```

<<*extension*>> (p. 1012) The original publication virtual sequence number.

If the **DDS_PresentationQosPolicy::access_scope** (p. 1893) of the **DDS_Publisher** (p. 443) is **DDS_GROUP_PPRESENTATION_QOS** (p. 1305), this field contains the **DDS_Publisher** (p. 443) virtual sequence number that uniquely identifies a sample within the DataWriter group.

See also

DDS_SampleInfo_get_sample_identity (p. 702)

6.176.6.18 related_original_publication_virtual_guid

```
struct DDS_GUID_t DDS_SampleInfo::related_original_publication_virtual_guid
```

<<**extension**>> (p. 1012) The original publication virtual GUID of a related sample.

See also

DDS_SampleInfo_get_related_sample_identity (p. 702)

6.176.6.19 related_original_publication_virtual_sequence_number

```
struct DDS_SequenceNumber_t DDS_SampleInfo::related_original_publication_virtual_sequence_number
```

<<**extension**>> (p. 1012) The original publication virtual sequence number of a related sample.

See also

DDS_SampleInfo_get_related_sample_identity (p. 702)

6.176.6.20 flag

```
DDS_SampleFlag DDS_SampleInfo::flag
```

<<**extension**>> (p. 1012) Flags associated with the sample.

The flags can be set by using the field **DDS_WriteParams_t::flag** (p. 2104) when writing a sample using the function **FooDataWriter_write_w_params** (p. 501).

6.176.6.21 source_guid

```
struct DDS_GUID_t DDS_SampleInfo::source_guid
```

<<**extension**>> (p. 1012) 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. 2104) when writing a sample using the function **FooDataWriter_write_w_params** (p. 501).

See also

DDS_WriteParams_t::source_guid (p. 2104) for additional information on **source_guid** (p. 1994) and when it is used.

6.176.6.22 related_source_guid

```
struct DDS_GUID_t DDS_SampleInfo::related_source_guid
```

<<**extension**>> (p. 1012) 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. 2105) when writing a sample using the function **FooDataWriter_write_w_params** (p. 501).

6.176.6.23 related_subscription_guid

```
struct DDS_GUID_t DDS_SampleInfo::related_subscription_guid
```

<<**extension**>> (p. 1012) 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. 2105) when writing a sample using the function **FooDataWriter_write_w_params** (p. 501).

6.176.6.24 topic_query_guid

```
struct DDS_GUID_t DDS_SampleInfo::topic_query_guid
```

<<**extension**>> (p. 1012) The GUID of the **DDS_TopicQuery** (p. 707) that is related to the sample.

This GUID indicates whether a sample is part of the response to a **DDS_TopicQuery** (p. 707) 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. 1213).

6.176.6.25 coherent_set_info

```
struct DDS_CoherentSetInfo_t* DDS_SampleInfo::coherent_set_info
```

<<**extension**>> (p. 1012) 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 **DDS_Publisher_begin_coherent_changes** (p. 459) and finalized using the operation **DDS_Publisher_end_coherent_changes** (p. 460).

See also

DDS_Publisher_begin_coherent_changes (p. 459) for additional information on coherent sets.

6.177 DDS_SampleInfoSeq Struct Reference

Declares IDL sequence < DDS_SampleInfo (p. 1985) > .

6.177.1 Detailed Description

Declares IDL sequence < DDS_SampleInfo (p. 1985) > .

See also

FooSeq (p. 2113)

Examples

HelloWorld_subscriber.c.

6.178 DDS_SampleLostStatus Struct Reference

DDS_SAMPLE_LOST_STATUS (p. 1230)

Data Fields

- **DDS_Long total_count**
*Total cumulative count of all samples lost across all instances of data published under the **DDS_Topic** (p. 175).*
- **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. 1012) Reason why the last sample was lost.

6.178.1 Detailed Description

DDS_SAMPLE_LOST_STATUS (p. 1230)

Examples

HelloWorld_subscriber.c.

6.178.2 Field Documentation

6.178.2.1 total_count

`DDS_Long DDS_SampleLostStatus::total_count`

Total cumulative count of all samples lost across all instances of data published under the **DDS_Topic** (p. 175).

6.178.2.2 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.

6.178.2.3 last_reason

`DDS_SampleLostStatusKind DDS_SampleLostStatus::last_reason`

<<*extension*>> (p. 1012) Reason why the last sample was lost.

See also

DDS_SampleLostStatusKind (p. 618)

6.179 DDS_SampleRejectedStatus Struct Reference

DDS_SAMPLE_REJECTED_STATUS (p. 1230)

Data Fields

- **DDS_Long total_count**
*Total cumulative count of samples rejected by the **DDS_DataReader** (p. 615).*
- **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.

6.179.1 Detailed Description

DDS_SAMPLE_REJECTED_STATUS (p. 1230)

Examples

`HelloWorld_subscriber.c.`

6.179.2 Field Documentation

6.179.2.1 `total_count`

`DDS_Long DDS_SampleRejectedStatus::total_count`

Total cumulative count of samples rejected by the **DDS_DataReader** (p. 615).

6.179.2.2 `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.

6.179.2.3 `last_reason`

`DDS_SampleRejectedStatusKind DDS_SampleRejectedStatus::last_reason`

Reason for rejecting the last sample rejected.

See also

DDS_SampleRejectedStatusKind (p. 622)

6.179.2.4 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. 624) and the **DDS_DataWriter** (p. 485) set **DDS_DataWriterProtocolQosPolicy::disable_inline_keyhash** (p. 1651) to **DDS_BOOLEAN_TRUE** (p. 1201), then the `last_instance_handle` may not be correct if the sample was encrypted.

6.180 DDS_SequenceNumber_t Struct Reference

Type for *sequence* number representation.

Data Fields

- **DDS_Long high**
The most significant part of the sequence number.
- **DDS_UnsignedLong low**
The least significant part of the sequence number.

6.180.1 Detailed Description

Type for *sequence* number representation.

Represents a 64-bit sequence number.

6.180.2 Field Documentation

6.180.2.1 high

`DDS_Long DDS_SequenceNumber_t::high`

The most significant part of the sequence number.

6.180.2.2 low

`DDS_UnsignedLong DDS_SequenceNumber_t::low`

The least significant part of the sequence number.

6.181 DDS_ServiceQosPolicy Struct Reference

<<*extension*>> (p. 795) Service or tool associated with a DDS entity.

Data Fields

- **DDS_ServiceQosPolicyKind** *kind*
<<*extension*>> (p. 795) *The kind of service.*

6.181.1 Detailed Description

<<*extension*>> (p. 795) 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:

DDS_DomainParticipant (p. 77), **DDS_DataReader** (p. 615), **DDS_DataWriter** (p. 485)

Properties:

RxO (p. 1245) = NO
Changeable (p. 1245) = **UNTIL ENABLE** (p. 1245)

6.181.2 Field Documentation

6.181.2.1 kind

DDS_ServiceQosPolicyKind `DDS_ServiceQosPolicy::kind`

<<*extension*>> (p. 795) The kind of service.

[default] **DDS_NO_SERVICE_QOS** (p. 1327)

Categories

Immutable (p. 914), **Discovery Metadata** (p. 958), **Debuggability** (p. 847)

6.182 DDS_ServiceRequest Struct Reference

A request coming from one of the built-in services.

Data Fields

- **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.

6.182.1 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. 1101). It contains service-specific information.

See also

DDS_SERVICE_REQUEST_TOPIC_NAME (p. 1101)
DDS_ParticipantBuiltinTopicDataDataReader (p. 1092)

6.182.2 Field Documentation

6.182.2.1 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. 1100)
DDS_TOPIC_QUERY_SERVICE_REQUEST_ID (p. 1101)

6.182.2.2 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.

6.182.2.3 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

DDS_TopicQueryHelper_topic_query_data_from_service_request (p. 708)

6.183 DDS_ServiceRequestAcceptedStatus Struct Reference

DDS_SERVICE_REQUEST_ACCEPTED_STATUS (p. 1233)

Data Fields

- **DDS_Long total_count**
*The total cumulative number of ServiceRequests that have been accepted by a **DDS_DataWriter** (p. 485).*
- **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 **DDS_DataWriter** (p. 485).*
- **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. 2001) that caused the **DDS_DataWriter** (p. 485)'s status to change.*
- **DDS_Long service_id**
ID of the service to which the accepted Request belongs.

6.183.1 Detailed Description

DDS_SERVICE_REQUEST_ACCEPTED_STATUS (p. 1233)

Currently, the only service that causes the ServiceRequestAcceptedStatus to be triggered is the **DDS_TopicQuery** (p. 707) service. A **DDS_ServiceRequest** (p. 2001) is accepted when a **DDS_DataWriter** (p. 485) matches with a **DDS_DataReader** (p. 615) that has created a **DDS_TopicQuery** (p. 707).

This status is also changed (and the listener, if any, called) when a ServiceRequest has been cancelled, or deleted. This will happen when a **DDS_DataReader** (p. 615) deletes a TopicQuery using **DDS_DataReader_delete_topic_query** (p. 692).

6.183.2 Field Documentation

6.183.2.1 total_count

DDS_Long DDS_ServiceRequestAcceptedStatus::total_count

The total cumulative number of ServiceRequests that have been accepted by a **DDS_DataWriter** (p. 485).

This number increases whenever a new request is accepted. It does not change when a request is cancelled.

6.183.2.2 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.

6.183.2.3 current_count

DDS_Long DDS_ServiceRequestAcceptedStatus::current_count

The current number of ServiceRequests that have been accepted by this **DDS_DataWriter** (p. 485).

This number increases when a new request is accepted and decreases when an existing request is cancelled.

6.183.2.4 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.

6.183.2.5 last_request_handle

`DDS_InstanceHandle_t DDS_ServiceRequestAcceptedStatus::last_request_handle`

A handle to the last **DDS_ServiceRequest** (p. 2001) that caused the **DDS_DataWriter** (p. 485)'s status to change.

6.183.2.6 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. 1101)

6.184 DDS_ServiceRequestSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_ServiceRequest** (p. 2001) > .

6.184.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_ServiceRequest** (p. 2001) > .

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_ServiceRequest (p. 2001)

6.185 DDS_ServiceRequestTypeSupport Struct Reference

Instantiates **TypeSupport** < **DDS_ServiceRequest** (p. 2001) > .

6.185.1 Detailed Description

Instantiates `TypeSupport` < `DDS_ServiceRequest` (p. 2001) > .

Instantiates:

<<*generic*>> (p. 1013) `FooTypeSupport` (p. 2114)

See also

`DDS_ServiceRequest` (p. 2001)

6.186 DDS_ShortSeq Struct Reference

Instantiates `FooSeq` (p. 2113) < `DDS_Short` (p. 1202) >

6.186.1 Detailed Description

Instantiates `FooSeq` (p. 2113) < `DDS_Short` (p. 1202) >

Instantiates:

<<*generic*>> (p. 1013) `FooSeq` (p. 2113)

See also

`DDS_Short` (p. 1202)

`FooSeq` (p. 2113)

6.187 DDS_StringSeq Struct Reference

Instantiates `FooSeq` (p. 2113) < `char*` > with value type semantics.

6.187.1 Detailed Description

Instantiates **FooSeq** (p. 2113) `< 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. 2005) 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. 2113) where **Foo** (p. 2109) is a pointer, ownership of **Foo** (p. 2109) implies ownership of `* Foo` (p. 2109). 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. 1521) 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. 2005) 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. 1526).

See also

String Support (p. 1526)

Instantiates:

`<<generic>>` (p. 1013) **FooSeq** (p. 2113)

See also

FooSeq (p. 2113)

6.188 DDS_StringTypeSupport Struct Reference

`<<interface>>` (p. 1013) String type support.

6.188.1 Detailed Description

<<*interface*>> (p. 1013) String type support.

6.189 DDS_StructMember Struct Reference

A description of a member of a struct.

Data Fields

- `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.

6.189.1 Detailed Description

A description of a member of a struct.

See also

`DDS_StructMemberSeq` (p. 2009)

`DDS_TypeCodeFactory_create_struct_tc` (p. 291)

6.189.2 Field Documentation

6.189.2.1 name

```
char* DDS_StructMember::name
```

The name of the struct member.

Cannot be NULL.

6.189.2.2 type

```
const DDS_TypeCode* DDS_StructMember::type
```

The type of the struct member.

Cannot be NULL.

6.189.2.3 is_pointer

```
DDS_Boolean DDS_StructMember::is_pointer
```

Indicates whether the struct member is a pointer or not.

6.189.2.4 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. 238).

6.189.2.5 is_key

```
DDS_Boolean DDS_StructMember::is_key
```

Indicates if the struct member is a key member or not.

6.189.2.6 id

```
DDS_Long DDS_StructMember::id
```

The member ID.

Use **DDS_TYPECODE_MEMBER_ID_INVALID** (p. 237) to have the member ID automatically assigned.

6.189.2.7 is_optional

```
DDS_Boolean DDS_StructMember::is_optional
```

Indicates if the struct member is optional or required.

6.190 DDS_StructMemberSeq Struct Reference

Defines a sequence of struct members.

6.190.1 Detailed Description

Defines a sequence of struct members.

See also

DDS_StructMember (p. 2007)

FooSeq (p. 2113)

DDS_TypeCodeFactory_create_struct_tc (p. 291)

6.191 DDS_SubscriberListener Struct Reference

<<*interface*>> (p. 1013) **DDS_Listener** (p. 1816) for status about a subscriber.

Data Fields

- struct **DDS_DataReaderListener as_datareaderlistener**
*The superclass instance of this **DDS_SubscriberListener** (p. 2009).*
- **DDS_SubscriberListener_DataOnReadersCallback on_data_on_readers**
*Handles the **DDS_DATA_ON_READERS_STATUS** (p. 1230) communication status.*

6.191.1 Detailed Description

<<*interface*>> (p. 1013) **DDS_Listener** (p. 1816) for status about a subscriber.

Entity:

DDS_Subscriber (p. 572)

Status:

DDS_DATA_AVAILABLE_STATUS (p. 1230);
DDS_DATA_ON_READERS_STATUS (p. 1230);
DDS_LIVELINESS_CHANGED_STATUS (p. 1231), **DDS_LivelinessChangedStatus** (p. 1820);
DDS_REQUESTED_DEADLINE_MISSED_STATUS (p. 1229), **DDS_RequestedDeadlineMissedStatus** (p. 1946);
DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS (p. 1229), **DDS_RequestedIncompatibleQosStatus** (p. 1947);
DDS_SAMPLE_LOST_STATUS (p. 1230), **DDS_SampleLostStatus** (p. 1996);
DDS_SAMPLE_REJECTED_STATUS (p. 1230), **DDS_SampleRejectedStatus** (p. 1997);
DDS_SUBSCRIPTION_MATCHED_STATUS (p. 1232), **DDS_SubscriptionMatchedStatus** (p. 2022);

See also

DDS_Listener (p. 1816)

Status Kinds (p. 1222)

Operations Allowed in Listener Callbacks (p. 1818)

6.191.2 Field Documentation

6.191.2.1 as_datareaderlistener

```
struct DDS_DataReaderListener DDS_SubscriberListener::as_datareaderlistener
```

The superclass instance of this **DDS_SubscriberListener** (p. 2009).

6.191.2.2 on_data_on_readers

```
DDS_SubscriberListener_DataOnReadersCallback DDS_SubscriberListener::on_data_on_readers
```

Handles the **DDS_DATA_ON_READERS_STATUS** (p. 1230) communication status.

6.192 DDS_SubscriberQos Struct Reference

QoS policies supported by a **DDS_Subscriber** (p. 572) entity.

Data Fields

- struct **DDS_PresentationQosPolicy** presentation
*Presentation policy, **PRESENTATION** (p. 1304).*
- struct **DDS_PartitionQosPolicy** partition
<<*basic*>> (p. 783) *Partition policy, **PARTITION** (p. 1303).*
- struct **DDS_GroupDataQosPolicy** group_data
*Group data policy, **GROUP_DATA** (p. 1293).*
- struct **DDS_EntityFactoryQosPolicy** entity_factory
<<*basic*>> (p. 783) *Entity factory policy, **ENTITY_FACTORY** (p. 1289).*
- struct **DDS_EntityNameQosPolicy** subscriber_name
<<*extension*>> (p. 1012) *EntityName policy, **ENTITY_NAME** (p. 1290).*

6.192.1 Detailed Description

QoS policies supported by a **DDS_Subscriber** (p. 572) entity.

You must set certain members in a consistent manner:

length of **DDS_GroupDataQosPolicy::value** (p. 1804) <= **DDS_DomainParticipantResourceLimitsQosPolicy::subscriber_group_data_max_length** (p. 1747)

length of **DDS_PartitionQosPolicy::name** (p. 1883) <= **DDS_DomainParticipantResourceLimitsQosPolicy::max_partitions** (p. 1748)

combined number of characters (including terminating 0) in **DDS_PartitionQosPolicy::name** (p. 1883) <= **DDS_DomainParticipantResourceLimitsQosPolicy::max_partition_cumulative_characters** (p. 1748)

If any of the above are not true, **DDS_Subscriber_set_qos** (p. 592) and **DDS_Subscriber_set_qos_with_profile** (p. 592) will fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222)

6.192.2 Field Documentation

6.192.2.1 presentation

```
struct DDS_PresentationQosPolicy DDS_SubscriberQos::presentation
```

Presentation policy, **PRESENTATION** (p. 1304).

6.192.2.2 partition

```
struct DDS_PartitionQosPolicy DDS_SubscriberQos::partition
```

<<*basic*>> (p. 783) Partition policy, **PARTITION** (p. 1303).

6.192.2.3 group_data

```
struct DDS_GroupDataQosPolicy DDS_SubscriberQos::group_data
```

Group data policy, **GROUP_DATA** (p. 1293).

6.192.2.4 entity_factory

```
struct DDS_EntityFactoryQosPolicy DDS_SubscriberQos::entity_factory
```

<<*basic*>> (p. 783) Entity factory policy, **ENTITY_FACTORY** (p. 1289).

6.192.2.5 subscriber_name

```
struct DDS_EntityNameQosPolicy DDS_SubscriberQos::subscriber_name
```

<<*extension*>> (p. 1012) EntityName policy, **ENTITY_NAME** (p. 1290).

6.193 DDS_SubscriberSeq Struct Reference

Declares IDL sequence < **DDS_Subscriber** (p. 572) > .

6.193.1 Detailed Description

Declares IDL sequence < **DDS_Subscriber** (p. 572) > .

See also

FooSeq (p. 2113)

6.194 DDS_SubscriptionBuiltinTopicData Struct Reference

Entry created when a **DDS_DataReader** (p. 615) is discovered in association with its Subscriber.

Data Fields

- **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 **DDS_Topic** (p. 175).*
- char * **type_name**
*Name of the type attached to the **DDS_Topic** (p. 175).*
- 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. 1012) Type code information of the corresponding Topic
- **DDS_BuiltinTopicKey_t subscriber_key**
 - <<extension>> (p. 1012) DCPS key of the subscriber to which the DataReader belongs.
- struct **DDS_PropertyQosPolicy property**
 - <<extension>> (p. 1012) Properties of the corresponding DataReader.
- struct **DDS_LocatorSeq unicast_locators**
 - <<extension>> (p. 1012) 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. 1012) 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. 1012) This field provides all the required information to enable content filtering on the Writer side.
- struct **DDS_GUID_t virtual_guid**
 - <<extension>> (p. 1012) Virtual GUID associated to the DataReader.
- struct **DDS_ServiceQosPolicy service**
 - <<extension>> (p. 1012) Policy of the corresponding DataReader.
- **DDS_ProtocolVersion_t rtps_protocol_version**
 - <<extension>> (p. 1012) Version number of the RTPS wire protocol used.
- struct **DDS_VendorId_t rtps_vendor_id**
 - <<extension>> (p. 1012) ID of vendor implementing the RTPS wire protocol.
- struct **DDS_ProductVersion_t product_version**
 - <<extension>> (p. 1012) This is a vendor specific parameter. It gives the current version of RTI Connex
- **DDS_Boolean disable_positive_acks**
 - <<extension>> (p. 1012) This is a vendor specific parameter. Determines whether the corresponding DataReader sends positive acknowledgments for reliability.
- struct **DDS_EntityNameQosPolicy subscription_name**
 - <<extension>> (p. 1012) The subscription name and role name.
- struct **DDS_EndpointTrustProtectionInfo trust_protection_info**
 - <<extension>> (p. 1012) Trust plugins protection information associated with the discovered DataReader.
- struct **DDS_EndpointTrustAlgorithmInfo trust_algorithm_info**
 - <<extension>> (p. 1012) Trust Plugins algorithms associated with the discovered DataReader.

- **DDS_BuiltinTopicKey_t related_datawriter_key**

<<*extension*>> (p. 1012) Key of the related *DataWriter* that is created as a part of the *Remote Procedure Call (RPC)* or *Request-Reply* endpoints.

6.194.1 Detailed Description

Entry created when a **DDS_DataReader** (p. 615) is discovered in association with its Subscriber.

Data associated with the built-in topic **DDS_SUBSCRIPTION_TOPIC_NAME** (p. 1098). It contains QoS policies and additional information that apply to the remote **DDS_DataReader** (p. 615) the related **DDS_Subscriber** (p. 572).

See also

DDS_SUBSCRIPTION_TOPIC_NAME (p. 1098)

DDS_SubscriptionBuiltinTopicDataDataReader (p. 1097)

6.194.2 Field Documentation

6.194.2.1 key

`DDS_BuiltinTopicKey_t DDS_SubscriptionBuiltinTopicData::key`

DCPS key to distinguish entries.

6.194.2.2 participant_key

`DDS_BuiltinTopicKey_t DDS_SubscriptionBuiltinTopicData::participant_key`

DCPS key of the participant to which the DataReader belongs.

6.194.2.3 topic_name

`char* DDS_SubscriptionBuiltinTopicData::topic_name`

Name of the related **DDS_Topic** (p. 175).

The length of this string is limited to 255 characters.

The memory for this field is managed as described in **String Conventions** (p. 1527).

See also

String Conventions (p. 1527)

6.194.2.4 type_name

```
char* DDS_SubscriptionBuiltinTopicData::type_name
```

Name of the type attached to the **DDS_Topic** (p. 175).

The length of this string is limited to 255 characters.

The memory for this field is managed as described in **String Conventions** (p. 1527).

See also

String Conventions (p. 1527)

6.194.2.5 durability

```
struct DDS_DurabilityQosPolicy DDS_SubscriptionBuiltinTopicData::durability
```

Policy of the corresponding DataReader.

6.194.2.6 deadline

```
struct DDS_DeadlineQosPolicy DDS_SubscriptionBuiltinTopicData::deadline
```

Policy of the corresponding DataReader.

6.194.2.7 latency_budget

```
struct DDS_LatencyBudgetQosPolicy DDS_SubscriptionBuiltinTopicData::latency_budget
```

Policy of the corresponding DataReader.

6.194.2.8 liveliness

```
struct DDS_LivelinessQosPolicy DDS_SubscriptionBuiltinTopicData::liveliness
```

Policy of the corresponding DataReader.

6.194.2.9 reliability

```
struct DDS_ReliabilityQosPolicy DDS_SubscriptionBuiltinTopicData::reliability
```

Policy of the corresponding DataReader.

6.194.2.10 ownership

```
struct DDS_OwnershipQosPolicy DDS_SubscriptionBuiltinTopicData::ownership
```

Policy of the corresponding DataReader.

6.194.2.11 destination_order

```
struct DDS_DestinationOrderQosPolicy DDS_SubscriptionBuiltinTopicData::destination_order
```

Policy of the corresponding DataReader.

Warning

Only the field **DDS_DestinationOrderQosPolicy::kind** (p. 1688) is propagated during discovery. The other fields always contain their default values.

6.194.2.12 user_data

```
struct DDS_UserDataQosPolicy DDS_SubscriptionBuiltinTopicData::user_data
```

Policy of the corresponding DataReader.

6.194.2.13 time_based_filter

```
struct DDS_TimeBasedFilterQosPolicy DDS_SubscriptionBuiltinTopicData::time_based_filter
```

Policy of the corresponding DataReader.

6.194.2.14 presentation

```
struct DDS_PresentationQosPolicy DDS_SubscriptionBuiltinTopicData::presentation
```

Policy of the Subscriber to which the DataReader belongs.

6.194.2.15 partition

```
struct DDS_PartitionQosPolicy DDS_SubscriptionBuiltinTopicData::partition
```

Policy of the Subscriber to which the DataReader belongs.

6.194.2.16 topic_data

```
struct DDS_TopicDataQosPolicy DDS_SubscriptionBuiltinTopicData::topic_data
```

Policy of the related Topic.

6.194.2.17 group_data

```
struct DDS_GroupDataQosPolicy DDS_SubscriptionBuiltinTopicData::group_data
```

Policy of the Subscriber to which the DataReader belongs.

6.194.2.18 type_consistency

```
struct DDS_TypeConsistencyEnforcementQosPolicy DDS_SubscriptionBuiltinTopicData::type_consistency
```

Policy of the corresponding DataReader.

6.194.2.19 representation

```
struct DDS_DataRepresentationQosPolicy DDS_SubscriptionBuiltinTopicData::representation
```

Data representation policy of the corresponding DataReader.

6.194.2.20 data_tags

`DDS_DataTagQosPolicy DDS_SubscriptionBuiltinTopicData::data_tags`

Tags of the corresponding DataReader.

6.194.2.21 type_code

`struct DDS_TypeCode* DDS_SubscriptionBuiltinTopicData::type_code`

<<*extension*>> (p. 1012) Type code information of the corresponding Topic

6.194.2.22 subscriber_key

`DDS_BuiltinTopicKey_t DDS_SubscriptionBuiltinTopicData::subscriber_key`

<<*extension*>> (p. 1012) DCPS key of the subscriber to which the DataReader belongs.

6.194.2.23 property

`struct DDS_PropertyQosPolicy DDS_SubscriptionBuiltinTopicData::property`

<<*extension*>> (p. 1012) Properties of the corresponding DataReader.

6.194.2.24 unicast_locators

`struct DDS_LocatorSeq DDS_SubscriptionBuiltinTopicData::unicast_locators`

<<*extension*>> (p. 1012) Custom unicast locators that the endpoint can specify. The default locators will be used if this is not specified.

6.194.2.25 multicast_locators

```
struct DDS_LocatorSeq DDS_SubscriptionBuiltinTopicData::multicast_locators
```

<<*extension*>> (p. 1012) Custom multicast locators that the endpoint can specify. The default locators will be used if this is not specified.

6.194.2.26 content_filter_property

```
struct DDS_ContentFilterProperty_t DDS_SubscriptionBuiltinTopicData::content_filter_property
```

<<*extension*>> (p. 1012) This field provides all the required information to enable content filtering on the Writer side.

6.194.2.27 virtual_guid

```
struct DDS_GUID_t DDS_SubscriptionBuiltinTopicData::virtual_guid
```

<<*extension*>> (p. 1012) Virtual GUID associated to the DataReader.

See also

DDS_GUID_t (p. 1804)

6.194.2.28 service

```
struct DDS_ServiceQosPolicy DDS_SubscriptionBuiltinTopicData::service
```

<<*extension*>> (p. 1012) Policy of the corresponding DataReader.

6.194.2.29 rtps_protocol_version

```
DDS_ProtocolVersion_t DDS_SubscriptionBuiltinTopicData::rtps_protocol_version
```

<<*extension*>> (p. 1012) Version number of the RTPS wire protocol used.

6.194.2.30 rtps_vendor_id

```
struct DDS_VendorId_t DDS_SubscriptionBuiltinTopicData::rtps_vendor_id
```

<<*extension*>> (p. 1012) ID of vendor implementing the RTPS wire protocol.

6.194.2.31 product_version

```
struct DDS_ProductVersion_t DDS_SubscriptionBuiltinTopicData::product_version
```

<<*extension*>> (p. 1012) This is a vendor specific parameter. It gives the current version of RTI Connext

6.194.2.32 disable_positive_acks

```
DDS_Boolean DDS_SubscriptionBuiltinTopicData::disable_positive_acks
```

<<*extension*>> (p. 1012) This is a vendor specific parameter. Determines whether the corresponding DataReader sends positive acknowledgments for reliability.

6.194.2.33 subscription_name

```
struct DDS_EntityNameQosPolicy DDS_SubscriptionBuiltinTopicData::subscription_name
```

<<*extension*>> (p. 1012) The subscription name and role name.

This member contains the name and the role name of the discovered subscription.

6.194.2.34 trust_protection_info

```
struct DDS_EndpointTrustProtectionInfo DDS_SubscriptionBuiltinTopicData::trust_protection_info
```

<<*extension*>> (p. 1012) 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](#).

6.194.2.35 trust_algorithm_info

```
struct DDS_EndpointTrustAlgorithmInfo DDS_SubscriptionBuiltinTopicData::trust_algorithm_info
```

<<**extension**>> (p. 1012) 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*.

6.194.2.36 related_datawriter_key

```
DDS_BuiltinTopicKey_t DDS_SubscriptionBuiltinTopicData::related_datawriter_key
```

<<**extension**>> (p. 1012) 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. 1111) only when the discovered DataReader is a part of Remote Procedure Call (RPC) or Request-Reply endpoints.

6.195 DDS_SubscriptionBuiltinTopicDataSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_SubscriptionBuiltinTopicData** (p. 2012) > .

6.195.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_SubscriptionBuiltinTopicData** (p. 2012) > .

Instantiates:

<<**generic**>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_SubscriptionBuiltinTopicData (p. 2012)

6.196 DDS_SubscriptionBuiltinTopicDataTypeSupport Struct Reference

Instantiates **TypeSupport** < **DDS_SubscriptionBuiltinTopicData** (p. 2012) > .

6.196.1 Detailed Description

Instantiates `TypeSupport < DDS_SubscriptionBuiltinTopicData` (p. 2012) > .

Instantiates:

`<<generic>>` (p. 1013) `FooTypeSupport` (p. 2114)

See also

`DDS_SubscriptionBuiltinTopicData` (p. 2012)

6.197 DDS_SubscriptionMatchedStatus Struct Reference

`DDS_SUBSCRIPTION_MATCHED_STATUS` (p. 1232)

Data Fields

- **DDS_Long total_count**
The total cumulative number of times that this `DDS_DataReader` (p. 615) discovered a "match" with a `DDS_DataWriter` (p. 485).
- **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 `DDS_DataReader` (p. 615) is matched.
- **DDS_Long current_count_peak**
`<<extension>>` (p. 1012) 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 `DDS_DataWriter` (p. 485) was the last to cause this `DataReader's` status to change, using `DDS_DataReader_get_matched_publication_data` (p. 679).

6.197.1 Detailed Description

`DDS_SUBSCRIPTION_MATCHED_STATUS` (p. 1232)

A "match" happens when the `DDS_DataReader` (p. 615) finds a `DDS_DataWriter` (p. 485) with the same `DDS_Topic` (p. 175), same or compatible data type, and an offered QoS that is compatible with that requested by the `DDS_DataReader` (p. 615). (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 `DDS_DataReader` (p. 615) will become "unmatched" from a remote `DDS_DataWriter` (p. 485) when that `DDS_DataWriter` (p. 485) goes away for any of the following reasons:

- The **DDS_DomainParticipant** (p. 77) containing the matched **DDS_DataWriter** (p. 485) 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 unmatched events, and the **DDS_SubscriptionMatchedStatus**↔
::current_count_change (p. 2024) 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. 1296)) does not trigger an unmatched event. So a DataWriter may still match even though its liveliness is lost.

Examples

HelloWorld_subscriber.c.

6.197.2 Field Documentation

6.197.2.1 total_count

DDS_Long DDS_SubscriptionMatchedStatus::total_count

The total cumulative number of times that this **DDS_DataReader** (p. 615) discovered a "match" with a **DDS_DataWriter** (p. 485).

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. 2022).

6.197.2.2 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).

6.197.2.3 current_count

DDS_Long DDS_SubscriptionMatchedStatus::current_count

The current number of DataWriters with which the **DDS_DataReader** (p. 615) 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. 2022).

6.197.2.4 `current_count_peak`

DDS_Long `DDS_SubscriptionMatchedStatus::current_count_peak`

<<*extension*>> (p. 1012) 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`.)

6.197.2.5 `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. 2022).

6.197.2.6 `last_publication_handle`

DDS_InstanceHandle_t `DDS_SubscriptionMatchedStatus::last_publication_handle`

This `InstanceHandle` can be used to look up which remote **DDS_DataWriter** (p. 485) was the last to cause this DataReader's status to change, using **DDS_DataReader_get_matched_publication_data** (p. 679).

If the DataWriter no longer matches this DataReader due to any of the reasons in **DDS_SubscriptionMatchedStatus** (p. 2022) 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 **DDS_DataReader_get_matched_publication_data** (p. 679) 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.

6.198 **DDS_SystemResourceLimitsQosPolicy** Struct Reference

<<*extension*>> (p. 795) Configures **DDS_DomainParticipant** (p. 77)-independent resources used by RTI Connext. Mainly used to change the maximum number of **DDS_DomainParticipant** (p. 77) entities that can be created within a single process (address space).

Data Fields

- **DDS_Long** `max_objects_per_thread`
[DEPRECATED] (p. 965) <<*extension*>> (p. 795) The maximum number of objects that can be stored per thread for a **DDS_DomainParticipantFactory** (p. 32).
- **DDS_Long** `initial_objects_per_thread`
<<*extension*>> (p. 795) The number of objects per thread for a **DDS_DomainParticipantFactory** (p. 32) for which infrastructure will initially be allocated.

6.198.1 Detailed Description

<<**extension**>> (p. 795) Configures **DDS_DomainParticipant** (p. 77)-independent resources used by RTI Connex. Mainly used to change the maximum number of **DDS_DomainParticipant** (p. 77) entities that can be created within a single process (address space).

Entity:

DDS_DomainParticipantFactory (p. 32)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

6.198.2 Usage

Within a single process (or address space for some supported real-time operating systems), applications may create and use multiple **DDS_DomainParticipant** (p. 77) entities. This QoS policy sets a parameter that places an effective upper bound on the maximum number of **DDS_DomainParticipant** (p. 77) 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.

6.198.3 Field Documentation

6.198.3.1 max_objects_per_thread

DDS_Long DDS_SystemResourceLimitsQosPolicy::max_objects_per_thread

[DEPRECATED] (p. 965) <<**extension**>> (p. 795) The maximum number of objects that can be stored per thread for a **DDS_DomainParticipantFactory** (p. 32).

This value is the upper bound on the number of objects that can be stored per thread. When a **DDS_DomainParticipantFactory** (p. 32) 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. 914), **Middleware Memory Management** (p. 878)

6.198.3.2 initial_objects_per_thread

DDS_Long DDS_SystemResourceLimitsQosPolicy::initial_objects_per_thread

<<**extension**>> (p. 795) The number of objects per thread for a **DDS_DomainParticipantFactory** (p. 32) 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. 914), **Middleware Memory Management** (p. 878)

6.199 DDS_Tag Struct Reference

Tags are name/value pair objects.

Data Fields

- char * **name**
Tag name.
- char * **value**
Tag value.

6.199.1 Detailed Description

Tags are name/value pair objects.

6.199.2 Field Documentation

6.199.2.1 name

```
char* DDS_Tag::name
```

Tag name.

It must be a NULL-terminated string.

6.199.2.2 value

```
char* DDS_Tag::value
```

Tag value.

It must be a NULL-terminated string.

6.200 DDS_TagSeq Struct Reference

Declares IDL sequence < **DDS_Tag** (p. 2026) >

6.200.1 Detailed Description

Declares IDL sequence < **DDS_Tag** (p. 2026) >

See also

DDS_Tag (p. 2026)

6.201 DDS_ThreadFactory Struct Reference

<<**extension**>> (p. 1012) <<**interface**>> (p. 1013) Interface for providing the threads needed by the middleware. It consists of operations to create and delete threads. The interface depends on the **DDS_ThreadFactory_OnSpawned**↔**Function** (p. 1433) that specifies the operation to run in the new thread.

Data Fields

- void * **factory_data**
A place for interface implementors to keep a pointer to data that may be needed by their factory.
- **DDS_ThreadFactory_CreateThreadCallback** **create_thread**
Handles the creation of new threads.
- **DDS_ThreadFactory_DeleteThreadCallback** **delete_thread**
Handles the deletion of threads previously created by this factory.

6.201.1 Detailed Description

<<*extension*>> (p. 1012) <<*interface*>> (p. 1013) Interface for providing the threads needed by the middleware. It consists of operations to create and delete threads. The interface depends on the **DDS_ThreadFactory_OnSpawned**↵
Function (p. 1433) that specifies the operation to run in the new thread.

6.201.2 Field Documentation

6.201.2.1 factory_data

```
void* DDS_ThreadFactory::factory_data
```

A place for interface implementors to keep a pointer to data that may be needed by their factory.

6.201.2.2 create_thread

```
DDS_ThreadFactory_CreateThreadCallback DDS_ThreadFactory::create_thread
```

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 **DDS_ThreadFactory_OnSpawnedFunction** (p. 1433). 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 **DDS_ThreadFactory_OnSpawnedFunction** (p. 1433), 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 **DDS_ThreadFactory_OnSpawnedFunction** (p. 1433), so the parameter can be passed directly:

```
int result = pthread_create(&pthread_handle, &attr, on_spawned, thread_params);
```

```
...
```

```
return &pthread_handle;
```

6.201.2.3 delete_thread

DDS_ThreadFactory_DeleteThreadCallback DDS_ThreadFactory::delete_thread

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 **DDS_ThreadFactory::create_thread** (p. 2028) 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().

6.202 DDS_ThreadSettings_t Struct Reference

The properties of a thread of execution.

Data Fields

- **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.

6.202.1 Detailed Description

The properties of a thread of execution.

QoS:

DDS_EventQosPolicy (p. 1793) **DDS_DatabaseQosPolicy** (p. 1581) **DDS_ReceiverPoolQosPolicy** (p. 1934)
DDS_AsynchronousPublisherQosPolicy (p. 1539)

6.202.2 Field Documentation

6.202.2.1 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. 1237).

[default] 0, use default options of the OS

6.202.2.2 priority

DDS_Long DDS_ThreadSettings_t::priority

Thread priority.

[range] Platform-dependent - Consult Platform Notes for additional details.

6.202.2.3 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. 878)

6.202.2.4 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 will 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. 1934). It is ignored within other QoS policies that include **DDS_ThreadSettings_t** (p. 2029).

See also

Controlling CPU Core Affinity for RTI Threads (p. 1237)

[default] Empty sequence

6.202.2.5 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. 1934). It is ignored within other QoS policies that include **DDS_ThreadSettings_t** (p. 2029).

See also

Controlling CPU Core Affinity for RTI Threads (p. 1237)

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. ;

6.203 DDS_Time_t Struct Reference

Type for *time* representation.

Data Fields

- **DDS_LongLong** *sec*
seconds
- **DDS_UnsignedLong** *nanosec*
nanoseconds

6.203.1 Detailed Description

Type for *time* representation.

A **DDS_Time_t** (p. 2031) represents a moment in time.

6.203.2 Field Documentation

6.203.2.1 sec

`DDS_LongLong DDS_Time_t::sec`

seconds

6.203.2.2 nanosec

`DDS_UnsignedLong DDS_Time_t::nanosec`

nanoseconds

[range] [0,1000000000)

6.204 DDS_TimeBasedFilterQosPolicy Struct Reference

<<*spec*>> (p. 790) Filter that allows a **DDS_DataReader** (p. 615) to specify that it is interested only in (potentially) a subset of the values of the data.

Data Fields

- struct **DDS_Duration_t** **minimum_separation**
 <<*basic*>> (p. 783) <<*spec*>> (p. 790) *The minimum separation duration between subsequent samples.*

6.204.1 Detailed Description

<<*spec*>> (p. 790) Filter that allows a **DDS_DataReader** (p. 615) to specify that it is interested only in (potentially) a subset of the values of the data.

The filter states that the **DDS_DataReader** (p. 615) does not want to receive more than one value each `minimum_separation`, regardless of how fast the changes occur.

Entity:

DDS_DataReader (p. 615)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **YES** (p. 1245)

6.204.2 Usage

You can use this QoS policy to reduce the amount of data received by a **DDS_DataReader** (p. 615). **DDS_DataWriter** (p. 485) entities may send data faster than needed by a **DDS_DataReader** (p. 615). For example, a **DDS_DataReader** (p. 615) 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 **DDS_DataWriter** (p. 485) of sensor information may have other **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615) entities can set their own time-based filters, so that data published faster than the period set by a each **DDS_DataReader** (p. 615) will not be delivered to that **DDS_DataReader** (p. 615).

The **TIME_BASED_FILTER** (p. 1329) also applies to each instance separately; that is, the constraint is that the **DDS_DataReader** (p. 615) 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 **DDS_DataReader** (p. 615), 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 **DDS_DataWriter** (p. 485) has an infinite **DDS_LivelinessQosPolicy::lease_duration** (p. 1825), RTI Connext will only send as many packets to a **DDS_DataReader** (p. 615) as required by the **TIME_BASED_FILTER**, no matter how fast **FooDataWriter_write** (p. 496) 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 **DDS_DataWriter** (p. 485) 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 **DDS_DataWriter** (p. 485) (for example, when the **DDS_DataWriter** (p. 485) is reliable, when the data type is keyed, or when the **DDS_DataWriter** (p. 485) has a finite **DDS_LivelinessQosPolicy::lease_duration** (p. 1825)), only the data that passes the **TIME_BASED_FILTER** will be stored in the receive queue of the **DDS_DataReader** (p. 615). 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).

6.204.3 Consistency

It is inconsistent for a **DDS_DataReader** (p. 615) to have a `minimum_separation` longer than its **DEADLINE** (p. 1272) `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. 1684) for more information about the interactions between deadlines and time-based filters.

The setting of a **TIME_BASED_FILTER** (p. 1329) – that is, the selection of a `minimum_separation` with a value greater than zero – is consistent with all settings of the **HISTORY** (p. 1292) and **RELIABILITY** (p. 1321) QoS. The **TIME_BASED_FILTER** (p. 1329) specifies the samples that are of interest to the **DDS_DataReader** (p. 615). The **HISTORY** (p. 1292) and **RELIABILITY** (p. 1321) QoS affect the behavior of the middleware with respect to the samples that have been determined to be of interest to the **DDS_DataReader** (p. 615); that is, they apply *after* the **TIME_BASED_FILTER** (p. 1329) has been applied.

In the case where the reliability QoS kind is **DDS_RELIABLE_RELIABILITY_QOS** (p. 1323), in steady-state – defined as the situation where the **DDS_DataWriter** (p. 485) 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 **DDS_DataReader** (p. 615).

See also

DeadlineQosPolicy
HistoryQosPolicy
ReliabilityQosPolicy

6.204.4 Field Documentation

6.204.4.1 `minimum_separation`

```
struct DDS_Duration_t DDS_TimeBasedFilterQosPolicy::minimum_separation
```

<<*basic*>> (p. 783) <<*spec*>> (p. 790) The minimum separation duration between subsequent samples.

[default] 0 (meaning the **DDS_DataReader** (p. 615) is potentially interested in all values)

[range] [0,1 year], < **DDS_DeadlineQosPolicy::period** (p. 1685)

Categories

Mutable (p. 949), **Discovery Metadata** (p. 958), **Applied Per-Instance** (p. 956), **Data Availability** (p. 843)

6.205 DDS_TopicBuiltinTopicData Struct Reference

Entry created when a Topic object discovered.

Data Fields

- **DDS_BuiltinTopicKey_t key**
DCPS key to distinguish entries.
- char * **name**
*Name of the **DDS_Topic** (p. 175).*
- char * **type_name**
*Name of the type attached to the **DDS_Topic** (p. 175).*
- 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.

6.205.1 Detailed Description

Entry created when a Topic object discovered.

Data associated with the built-in topic **DDS_TOPIC_TOPIC_NAME** (p. 1094). It contains QoS policies and additional information that apply to the remote **DDS_Topic** (p. 175).

Note: The **DDS_TopicBuiltinTopicData** (p. 2034) 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. 1905) and **DDS_SubscriptionBuiltinTopicData** (p. 2012)). Therefore TopicBuiltinTopicData DataReaders will not receive any data.

See also

DDS_TOPIC_TOPIC_NAME (p. 1094)

DDS_TopicBuiltinTopicDataDataReader (p. 1093)

6.205.2 Field Documentation

6.205.2.1 key

`DDS_BuiltinTopicKey_t DDS_TopicBuiltinTopicData::key`

DCPS key to distinguish entries.

6.205.2.2 name

`char* DDS_TopicBuiltinTopicData::name`

Name of the **DDS_Topic** (p. 175).

The length of this string is limited to 255 characters.

The memory for this field is managed as described in **String Conventions** (p. 1527).

See also

String Conventions (p. 1527)

6.205.2.3 type_name

`char* DDS_TopicBuiltinTopicData::type_name`

Name of the type attached to the **DDS_Topic** (p. 175).

The length of this string is limited to 255 characters.

The memory for this field is managed as described in **String Conventions** (p. 1527).

See also

String Conventions (p. 1527)

6.205.2.4 durability

```
struct DDS_DurabilityQosPolicy DDS_TopicBuiltinTopicData::durability
```

durability policy of the corresponding Topic

6.205.2.5 durability_service

```
struct DDS_DurabilityServiceQosPolicy DDS_TopicBuiltinTopicData::durability_service
```

durability service policy of the corresponding Topic

6.205.2.6 deadline

```
struct DDS_DeadlineQosPolicy DDS_TopicBuiltinTopicData::deadline
```

Policy of the corresponding Topic.

6.205.2.7 latency_budget

```
struct DDS_LatencyBudgetQosPolicy DDS_TopicBuiltinTopicData::latency_budget
```

Policy of the corresponding Topic.

6.205.2.8 liveliness

```
struct DDS_LivelinessQosPolicy DDS_TopicBuiltinTopicData::liveliness
```

Policy of the corresponding Topic.

6.205.2.9 reliability

```
struct DDS_ReliabilityQosPolicy DDS_TopicBuiltinTopicData::reliability
```

Policy of the corresponding Topic.

6.205.2.10 transport_priority

```
struct DDS_TransportPriorityQosPolicy DDS_TopicBuiltinTopicData::transport_priority
```

Policy of the corresponding Topic.

6.205.2.11 lifespan

```
struct DDS_LifespanQosPolicy DDS_TopicBuiltinTopicData::lifespan
```

Policy of the corresponding Topic.

6.205.2.12 destination_order

```
struct DDS_DestinationOrderQosPolicy DDS_TopicBuiltinTopicData::destination_order
```

Policy of the corresponding Topic.

6.205.2.13 history

```
struct DDS_HistoryQosPolicy DDS_TopicBuiltinTopicData::history
```

Policy of the corresponding Topic.

6.205.2.14 resource_limits

```
struct DDS_ResourceLimitsQosPolicy DDS_TopicBuiltinTopicData::resource_limits
```

Policy of the corresponding Topic.

6.205.2.15 ownership

```
struct DDS_OwnershipQosPolicy DDS_TopicBuiltinTopicData::ownership
```

Policy of the corresponding Topic.

6.205.2.16 topic_data

```
struct DDS_TopicDataQosPolicy DDS_TopicBuiltinTopicData::topic_data
```

Policy of the corresponding Topic.

6.205.2.17 representation

```
struct DDS_DataRepresentationQosPolicy DDS_TopicBuiltinTopicData::representation
```

Data representation policy of the corresponding Topic.

6.206 DDS_TopicBuiltinTopicDataSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_TopicBuiltinTopicData** (p. 2034) > .

6.206.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_TopicBuiltinTopicData** (p. 2034) > .

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_TopicBuiltinTopicData (p. 2034)

6.207 DDS_TopicBuiltinTopicDataTypeSupport Struct Reference

Instantiates **TypeSupport** < **DDS_TopicBuiltinTopicData** (p. 2034) > .

6.207.1 Detailed Description

Instantiates **TypeSupport** < **DDS_TopicBuiltinTopicData** (p. 2034) > .

Instantiates:

<<*generic*>> (p. 1013) **FooTypeSupport** (p. 2114)

See also

DDS_TopicBuiltinTopicData (p. 2034)

6.208 DDS_TopicDataQosPolicy Struct Reference

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 165) during discovery.

Data Fields

- struct **DDS_OctetSeq** value
a sequence of octets

6.208.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 165) during discovery.

Entity:

DDS_Topic (p. 175)

Properties:

RxO (p. 1245) = NO

Changeable (p. 1245) = YES (p. 1245)

See also

DDS_DomainParticipant_get_builtin_subscriber (p. 129)

6.208.2 Usage

The purpose of this QoS is to allow the application to attach additional information to the created **DDS_Topic** (p. 175) 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 Connex.

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 **DDS_DataReaderListener** (p. 1593), **DDS_DataWriterListener** (p. 1644), or operations such as **DDS_DomainParticipant_ignore_topic** (p. 132), 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 Connex stores the data placed in this policy in pre-allocated pools. It is therefore necessary to configure RTI Connex 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. 1746).

6.208.3 Field Documentation

6.208.3.1 value

```
struct DDS_OctetSeq DDS_TopicDataQosPolicy::value
```

a sequence of octets

[default] empty (zero-length)

[range] Octet sequence of length [0,max_length]

6.209 DDS_Listener Struct Reference

<<*interface*>> (p. 1013) **DDS_Listener** (p. 1816) for **DDS_Topic** (p. 175) entities.

Data Fields

- struct **DDS_Listener** as **listener**
*The superclass instance of this **DDS_TopicListener** (p. 2041).*
- **DDS_TopicListener_InconsistentTopicCallback** **on_inconsistent_topic**
*Handle the **DDS_INCONSISTENT_TOPIC_STATUS** (p. 1228) status.*

6.209.1 Detailed Description

<<*interface*>> (p. 1013) **DDS_Listener** (p. 1816) for **DDS_Topic** (p. 175) entities.

Entity:

DDS_Topic (p. 175)

Status:

DDS_INCONSISTENT_TOPIC_STATUS (p. 1228), **DDS_InconsistentTopicStatus** (p. 1808)

This is the interface that can be implemented by an application-provided class and then registered with the **DDS_Topic** (p. 175) such that the application can be notified by RTI Connex of relevant status changes.

See also

- Status Kinds** (p. 1222)
- DDS_Listener** (p. 1816)
- DDS_Topic_set_listener** (p. 199)
- Operations Allowed in Listener Callbacks** (p. 1818)

6.209.2 Field Documentation

6.209.2.1 as_listener

```
struct DDS_Listener DDS_TopicListener::as_listener
```

The superclass instance of this **DDS_TopicListener** (p. 2041).

6.209.2.2 on_inconsistent_topic

```
DDS_TopicListener_InconsistentTopicCallback DDS_TopicListener::on_inconsistent_topic
```

Handle the **DDS_INCONSISTENT_TOPIC_STATUS** (p. 1228) status.

This callback is called when a remote **DDS_Topic** (p. 175) is discovered but is inconsistent with the locally created **DDS_Topic** (p. 175) of the same topic name.

6.210 DDS_TopicQos Struct Reference

QoS policies supported by a **DDS_Topic** (p. 175) entity.

Data Fields

- struct **DDS_TopicDataQosPolicy** **topic_data**
*Topic data policy, **TOPIC_DATA** (p. 1329).*
- struct **DDS_DurabilityQosPolicy** **durability**
*Durability policy, **DURABILITY** (p. 1285).*
- struct **DDS_DurabilityServiceQosPolicy** **durability_service**
*DurabilityService policy, **DURABILITY_SERVICE** (p. 1289).*
- struct **DDS_DeadlineQosPolicy** **deadline**
*<<basic>> (p. 783) Deadline policy, **DEADLINE** (p. 1272).*
- struct **DDS_LatencyBudgetQosPolicy** **latency_budget**
*Latency budget policy, **LATENCY_BUDGET** (p. 1294).*
- struct **DDS_LivelinessQosPolicy** **liveliness**
*Liveliness policy, **LIVELINESS** (p. 1296).*
- struct **DDS_ReliabilityQosPolicy** **reliability**
*Reliability policy, **RELIABILITY** (p. 1321).*
- struct **DDS_DestinationOrderQosPolicy** **destination_order**
*Destination order policy, **DESTINATION_ORDER** (p. 1273).*
- struct **DDS_HistoryQosPolicy** **history**

- `<<basic>>` (p. 783) History policy, **HISTORY** (p. 1292).
- struct **DDS_ResourceLimitsQosPolicy** `resource_limits`
Resource limits policy, **RESOURCE_LIMITS** (p. 1325).
- struct **DDS_TransportPriorityQosPolicy** `transport_priority`
Transport priority policy, **TRANSPORT_PRIORITY** (p. 1337).
- struct **DDS_LifespanQosPolicy** `lifespan`
`<<basic>>` (p. 783) Lifespan policy, **LIFESPAN** (p. 1295).
- struct **DDS_OwnershipQosPolicy** `ownership`
`<<basic>>` (p. 783) Ownership policy, **OWNERSHIP** (p. 1301).
- struct **DDS_DataRepresentationQosPolicy** `representation`
Data representation policy, **DATA_REPRESENTATION** (p. 1256).

6.210.1 Detailed Description

QoS policies supported by a **DDS_Topic** (p. 175) entity.

You must set certain members in a consistent manner:

length of **DDS_TopicQos::topic_data** (p. 2043) .value <= **DDS_DomainParticipantQos::resource_limits** (p. 1729) .topic_data_max_length

If any of the above are not true, **DDS_Topic_set_qos** (p. 197), **DDS_Topic_set_qos_with_profile** (p. 198) and **DDS_DomainParticipant_set_default_topic_qos** (p. 87) will fail with **DDS_RETCODE_INCONSISTENT_POLICY** (p. 1222) and **DDS_DomainParticipant_create_topic** (p. 115) will return NULL.

Entity:

DDS_Topic (p. 175)

See also

All QoS Policies (p. 1238) allowed ranges within each Qos.

6.210.2 Field Documentation

6.210.2.1 topic_data

```
struct DDS_TopicDataQosPolicy DDS_TopicQos::topic_data
```

Topic data policy, **TOPIC_DATA** (p. 1329).

6.210.2.2 durability

```
struct DDS_DurabilityQosPolicy DDS_TopicQos::durability
```

Durability policy, **DURABILITY** (p. 1285).

6.210.2.3 durability_service

```
struct DDS_DurabilityServiceQosPolicy DDS_TopicQos::durability_service
```

DurabilityService policy, **DURABILITY_SERVICE** (p. 1289).

6.210.2.4 deadline

```
struct DDS_DeadlineQosPolicy DDS_TopicQos::deadline
```

<<**basic**>> (p. 783) Deadline policy, **DEADLINE** (p. 1272).

6.210.2.5 latency_budget

```
struct DDS_LatencyBudgetQosPolicy DDS_TopicQos::latency_budget
```

Latency budget policy, **LATENCY_BUDGET** (p. 1294).

6.210.2.6 liveliness

```
struct DDS_LivelinessQosPolicy DDS_TopicQos::liveliness
```

Liveliness policy, **LIVELINESS** (p. 1296).

6.210.2.7 reliability

```
struct DDS_ReliabilityQosPolicy DDS_TopicQos::reliability
```

Reliability policy, **RELIABILITY** (p. 1321).

6.210.2.8 destination_order

```
struct DDS_DestinationOrderQosPolicy DDS_TopicQos::destination_order
```

Destination order policy, **DESTINATION_ORDER** (p. 1273).

6.210.2.9 history

```
struct DDS_HistoryQosPolicy DDS_TopicQos::history
```

<<*basic*>> (p. 783) History policy, **HISTORY** (p. 1292).

6.210.2.10 resource_limits

```
struct DDS_ResourceLimitsQosPolicy DDS_TopicQos::resource_limits
```

Resource limits policy, **RESOURCE_LIMITS** (p. 1325).

6.210.2.11 transport_priority

```
struct DDS_TransportPriorityQosPolicy DDS_TopicQos::transport_priority
```

Transport priority policy, **TRANSPORT_PRIORITY** (p. 1337).

6.210.2.12 lifespan

```
struct DDS_LifespanQosPolicy DDS_TopicQos::lifespan
```

<<*basic*>> (p. 783) Lifespan policy, **LIFESPAN** (p. 1295).

6.210.2.13 ownership

```
struct DDS_OwnershipQosPolicy DDS_TopicQos::ownership
```

<<*basic*>> (p. 783) Ownership policy, **OWNERSHIP** (p. 1301).

6.210.2.14 representation

```
struct DDS_DataRepresentationQosPolicy DDS_TopicQos::representation
```

Data representation policy, **DATA_REPRESENTATION** (p. 1256).

6.211 DDS_TopicQueryData Struct Reference

<<**extension**>> (p. 1012) Provides information about a **DDS_TopicQuery** (p. 707)

Data Fields

- struct **DDS_TopicQuerySelection** **topic_query_selection**
The data selection.
- char * **topic_name**
*The topic name of the **DDS_DataReader** (p. 615).*
- struct **DDS_GUID_t** **original_related_reader_guid**
*Identifies the **DDS_DataReader** (p. 615) that created the **DDS_TopicQuery** (p. 707).*

6.211.1 Detailed Description

<<**extension**>> (p. 1012) Provides information about a **DDS_TopicQuery** (p. 707)

Contains the information about a TopicQuery that can be retrieved using **DDS_TopicQueryHelper_topic_query_↔data_from_service_request** (p. 708).

See also

DDS_TopicQueryHelper_topic_query_data_from_service_request (p. 708)

6.211.2 Field Documentation

6.211.2.1 topic_query_selection

```
struct DDS_TopicQuerySelection DDS_TopicQueryData::topic_query_selection
```

The data selection.

6.211.2.2 topic_name

```
char* DDS_TopicQueryData::topic_name
```

The topic name of the **DDS_DataReader** (p. 615).

6.211.2.3 original_related_reader_guid

```
struct DDS_GUID_t DDS_TopicQueryData::original_related_reader_guid
```

Identifies the **DDS_DataReader** (p. 615) that created the **DDS_TopicQuery** (p. 707).

6.212 DDS_TopicQueryDispatchQosPolicy Struct Reference

<<**extension**>> (p. 795) Configures the ability of a **DDS_DataWriter** (p. 485) to publish samples in response to a **DDS_TopicQuery** (p. 707)

Data Fields

- **DDS_Boolean enable**
 <<**extension**>> (p. 795) *Allows this writer to dispatch TopicQueries*
- struct **DDS_Duration_t publication_period**
 <<**basic**>> (p. 783) <<**extension**>> (p. 795) *Sets the periodic interval at which samples are published.*
- **DDS_Long samples_per_period**
 <<**extension**>> (p. 795) *Sets the maximum number of samples to publish in each publication_period.*

6.212.1 Detailed Description

<<**extension**>> (p. 795) Configures the ability of a **DDS_DataWriter** (p. 485) to publish samples in response to a **DDS_TopicQuery** (p. 707)

Enables the ability of a **DDS_DataWriter** (p. 485) to publish historical samples upon reception of a **DDS_TopicQuery** (p. 707) and how often they are published.

Since a TopicQuery selects previously written samples, the DataWriter must have a **DDS_DurabilityQosPolicy**↔
::kind (p. 1762) different from **DDS_VOLATILE_DURABILITY_QOS** (p. 1287). Also, **DDS_ReliabilityQosPolicy::kind**
 (p. 1939) must be set to **DDS_RELIABLE_RELIABILITY_QOS** (p. 1323).

Only samples that fall within the **DDS_DurabilityQosPolicy::writer_depth** (p. 1762) 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. 1762) samples per instance in the

DataWriter's queue if the **DDS_HistoryQosPolicy::depth** (p. 1807) is set to a higher value than **DDS_DurabilityQosPolicy::writer_depth** (p. 1762). Those additional samples past **DDS_DurabilityQosPolicy::writer_depth** (p. 1762) 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. 2048) configures the frequency of that period and **DDS_TopicQueryDispatchQosPolicy::samples_per_period** (p. 2049) 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. 496) for the conditions that may cause the write operation to block.)

All the DataWriters that belong to a single **DDS_Publisher** (p. 443) and enable TopicQueries share the same event thread, but each DataWriter schedules separate events. To configure that thread, see **DDS_AsymynchronousPublisherQosPolicy::topic_query_publication_thread** (p. 1542).

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:

DDS_DataWriter (p. 485)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

See also

DDS_Publisher (p. 443)

6.212.2 Field Documentation

6.212.2.1 enable

DDS_Boolean **DDS_TopicQueryDispatchQosPolicy::enable**

<<**extension**>> (p. 795) Allows this writer to dispatch TopicQueries

When set to true, this writer can receive and dispatch TopicQueries.

[default] **DDS_BOOLEAN_FALSE** (p. 1201)

Categories

Topic Queries (p. 911), **Immutable** (p. 914), **Discovery Metadata** (p. 958)

6.212.2.2 publication_period

```
struct DDS_Duration_t DDS_TopicQueryDispatchQosPolicy::publication_period
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Sets the periodic interval at which samples are published.

[default] 1 second

[range] [0,1 year]

Categories

Topic Queries (p. 911), Immutable (p. 914)

6.212.2.3 samples_per_period

```
DDS_Long DDS_TopicQueryDispatchQosPolicy::samples_per_period
```

<<**extension**>> (p. 795) Sets the maximum number of samples to publish in each publication_period.

[default] DDS_LENGTH_UNLIMITED (p. 1326)

[range] [1, 100000000] or DDS_LENGTH_UNLIMITED (p. 1326)

Categories

Topic Queries (p. 911), Immutable (p. 914)

6.213 DDS_TopicQuerySelection Struct Reference

<<**extension**>> (p. 1012) Specifies the data query that defines a **DDS_TopicQuery** (p. 707)

Data Fields

- 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.

6.213.1 Detailed Description

<<**extension**>> (p. 1012) Specifies the data query that defines a **DDS_TopicQuery** (p. 707)

The query format is similar to the expression and parameters of a **DDS_QueryCondition** (p. 699) or a **DDS_ContentFilteredTopic** (p. 177), as described in **DDS_DomainParticipant_create_contentfilteredtopic_with_filter** (p. 119).

There are two special queries:

- **DDS_TOPIC_QUERY_SELECTION_SELECT_ALL** (p. 709)
- **DDS_TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER** (p. 709)

See also

Queries and Filters Syntax (p. 738)

6.213.2 Field Documentation

6.213.2.1 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 **DDS_DomainParticipant_register_contentfilter** (p. 120) on the same **DDS_DomainParticipant** (p. 77).

Built-in filter names are **DDS_SQLFILTER_NAME** (p. 163) and **DDS_STRINGMATCHFILTER_NAME** (p. 164).

[default] See **DDS_TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER** (p. 709)

6.213.2.2 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. 738).

[default] See **DDS_TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER** (p. 709)

See also

Queries and Filters Syntax (p. 738)

6.213.2.3 filter_parameters

```
struct DDS_StringSeq DDS_TopicQuerySelection::filter_parameters
```

The query parameters.

[default] See `DDS_TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER` (p. 709)

See also

`DDS_DomainParticipant_create_contentfilteredtopic_with_filter` (p. 119).

6.213.2.4 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. 708)

6.214 DDS_TransportBuiltinQosPolicy Struct Reference

<<*extension*>> (p. 795) Specifies which built-in transports are used.

Data Fields

- `DDS_TransportBuiltinKindMask mask`

<<*basic*>> (p. 783) <<*extension*>> (p. 795) Specifies the built-in transports that are registered automatically when the `DDS_DomainParticipant` (p. 77) is enabled.

6.214.1 Detailed Description

<<*extension*>> (p. 795) 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 `DDS_DomainParticipant` (p. 77). 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:

`DDS_DomainParticipant` (p. 77)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

6.214.2 Field Documentation

6.214.2.1 mask

`DDS_TransportBuiltinKindMask DDS_TransportBuiltinQosPolicy::mask`

<<**basic**>> (p. 783) <<**extension**>> (p. 795) Specifies the built-in transports that are registered automatically when the **DDS_DomainParticipant** (p. 77) is enabled.

RTI Connext provides several built-in transports. Only those that are specified with this mask are registered automatically when the **DDS_DomainParticipant** (p. 77) is enabled.

[default] **DDS_TRANSPORTBUILTIN_MASK_DEFAULT** (p. 1332)

Categories

Immutable (p. 914), **Transports** (p. 908)

6.215 DDS_TransportInfo_t Struct Reference

Contains the `class_id` and `message_size_max` of an installed transport.

Data Fields

- **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.

6.215.1 Detailed Description

Contains the `class_id` and `message_size_max` of an installed transport.

6.215.2 Field Documentation

6.215.2.1 class_id

`NDDS_Transport_ClassId_t DDS_TransportInfo_t::class_id`

The `class_id` identifies the transport associated with the `message_size_max`.

6.215.2.2 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`.

6.216 DDS_TransportInfoSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_TransportInfo_t** (p. 2052) > .

6.216.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_TransportInfo_t** (p. 2052) > .

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_TransportInfo_t (p. 2052)

6.217 DDS_TransportMulticastMapping_t Struct Reference

<<*extension*>> (p. 795) Type representing a list of multicast mapping elements.

Data Fields

- `char * addresses`
 <<*extension*>> (p. 795) 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. 2054).
- `char * topic_expression`
 <<*extension*>> (p. 795) A regular expression that will be used to map topic names to corresponding multicast receive addresses.
- `struct DDS_TransportMulticastMappingFunction_t mapping_function`
 <<*extension*>> (p. 795) Specifies a function that will define the mapping between a topic name and a specific multicast address from a list of addresses.

6.217.1 Detailed Description

<<**extension**>> (p. 795) 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. 2056)

6.217.2 Field Documentation

6.217.2.1 addresses

```
char* DDS_TransportMulticastMapping_t::addresses
```

<<**extension**>> (p. 795) 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. 2054).

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

6.217.2.2 topic_expression

```
char* DDS_TransportMulticastMapping_t::topic_expression
```

<<**extension**>> (p. 795) 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

6.217.2.3 mapping_function

```
struct DDS_TransportMulticastMappingFunction_t DDS_TransportMulticastMapping_t::mapping_function
```

<<**extension**>> (p. 795) 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.

6.218 DDS_TransportMulticastMappingFunction_t Struct Reference

Type representing an external mapping function.

Data Fields

- char * **dll**
Specifies a dynamic library that contains a mapping function.
- char * **function_name**
Specifies the name of a mapping function.

6.218.1 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. 2056)

6.218.2 Field Documentation

6.218.2.1 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

6.218.2.2 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. 2055).

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.

6.219 DDS_TransportMulticastMappingQosPolicy Struct Reference

<<**extension**>> (p. 795) Specifies a list of `topic_expressions` and multicast addresses that can be used by an Entity with a specific topic name to receive data.

Data Fields

- struct **DDS_TransportMulticastMappingSeq** **value**
 <<**extension**>> (p. 795) *A sequence of multicast communication mappings.*

6.219.1 Detailed Description

<<**extension**>> (p. 795) 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 **DDS_DomainParticipant** (p. 77) level.

To use multicast communication *without* this QoS policy, you must explicitly assign a multicast receive address on each **DDS_DataReader** (p. 615). 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 **DDS_DomainParticipant** (p. 77); those addresses will then be automatically assigned to the DomainParticipant's DataReaders. A single configuration on the **DDS_DomainParticipant** (p. 77) 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. 2059) to **DDS_AUTOMATIC_TRANSPORT_MULTICAST_QOS** (p. 1336).
- Set the first element in **DDS_DataReader** (p. 615)'s **DDS_TransportMulticastQosPolicy::value** (p. 2059) with an empty address.

Entity:

DDS_DomainParticipant (p. 77)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

6.219.2 Field Documentation

6.219.2.1 value

struct **DDS_TransportMulticastMappingSeq** DDS_TransportMulticastMappingQosPolicy::value

<<*extension*>> (p. 795) A sequence of multicast communication mappings.

[**default**] Empty sequence.

Categories

Immutable (p. 914), **Transports** (p. 908), **Data Availability** (p. 843)

6.220 DDS_TransportMulticastMappingSeq Struct Reference

Declares IDL sequence< **DDS_TransportMulticastMapping_t** (p. 2053) >

6.220.1 Detailed Description

Declares IDL sequence< **DDS_TransportMulticastMapping_t** (p. 2053) >

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_TransportMulticastMapping_t (p. 2053)

6.221 DDS_TransportMulticastQosPolicy Struct Reference

<<*extension*>> (p. 795) Specifies the multicast address on which a **DDS_DataReader** (p. 615) wants to receive its data. It can also specify a port number as well as a subset of the available (at the **DDS_DomainParticipant** (p. 77) level) transports with which to receive the multicast data.

Data Fields

- struct **DDS_TransportMulticastSettingsSeq** value
 <<*basic*>> (p. 783) <<*extension*>> (p. 795) A sequence of multicast communications settings.
- **DDS_TransportMulticastQosPolicyKind** kind
 <<*extension*>> (p. 795) A value that specifies a way to determine how to obtain the multicast address.

6.221.1 Detailed Description

<<**extension**>> (p. 795) Specifies the multicast address on which a **DDS_DataReader** (p. 615) wants to receive its data. It can also specify a port number as well as a subset of the available (at the **DDS_DomainParticipant** (p. 77) level) transports with which to receive the multicast data.

By default, a **DDS_DataWriter** (p. 485) will send individually addressed packets for each **DDS_DataReader** (p. 615) 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 **DDS_DataWriter** (p. 485) 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 **DDS_Topic** (p. 175).

Entity:

DDS_DataReader (p. 615)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

6.221.2 Field Documentation

6.221.2.1 value

```
struct DDS_TransportMulticastSettingsSeq DDS_TransportMulticastQosPolicy::value
```

<<**basic**>> (p. 783) <<**extension**>> (p. 795) 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. 1901) associated with the **DDS_DomainParticipantQos** (p. 1726).

[**default**] Empty sequence.

Categories

Immutable (p. 914), **Transports** (p. 908), **Discovery Metadata** (p. 958)

6.221.2.2 kind

`DDS_TransportMulticastQosPolicyKind DDS_TransportMulticastQosPolicy::kind`

<<*extension*>> (p. 795) 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. 1336) or **DDS_UNICAST_ONLY_TRANSPORT_MULTICAST_QOS** (p. 1336).

- If it is set to **DDS_AUTOMATIC_TRANSPORT_MULTICAST_QOS** (p. 1336), the behavior will depend on the **DDS_TransportMulticastQosPolicy::value** (p. 2059):
 - If **DDS_TransportMulticastQosPolicy::value** (p. 2059) does not have any elements, then multicast will not be used.
 - If **DDS_TransportMulticastQosPolicy::value** (p. 2059) first element has an empty address, then the address will be obtained from **DDS_TransportMulticastMappingQosPolicy** (p. 2056).
 - If none of the elements in **DDS_TransportMulticastQosPolicy::value** (p. 2059) 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. 1336), then multicast will not be used.

[default] **DDS_AUTOMATIC_TRANSPORT_MULTICAST_QOS** (p. 1336)

Categories

Immutable (p. 914), **Transports** (p. 908), **Data Availability** (p. 843)

6.222 DDS_TransportMulticastSettings_t Struct Reference

Type representing a list of multicast locators.

Data Fields

- struct **DDS_StringSeq transports**
 - <<*extension*>> (p. 795) A sequence of transport aliases that specifies the transports on which to receive multicast traffic for the entity.
- char * **receive_address**
 - <<*extension*>> (p. 795) The multicast group address on which the entity can receive data.
- **DDS_Long receive_port**
 - <<*extension*>> (p. 795) The multicast port on which the entity can receive data.

6.222.1 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. 2058)

6.222.2 Field Documentation

6.222.2.1 transports

```
struct DDS_StringSeq DDS_TransportMulticastSettings_t::transports
```

<<**extension**>> (p. 795) 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. 1527). In particular, be careful to avoid a situation in which RTI Connexx allocates a string on your behalf and you then reuse that string in such a way that RTI Connexx 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. 1331).

[default] Empty sequence; i.e. all the transports available to the entity.

[range] Any sequence of non-null, non-empty strings.

6.222.2.2 receive_address

```
char* DDS_TransportMulticastSettings_t::receive_address
```

<<**extension**>> (p. 795) The multicast group address on which the entity can receive data.

Must must be an address in the proper format (see **Address Format** (p. 1354)).

[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. 1354)

6.222.2.3 receive_port

DDS_Long DDS_TransportMulticastSettings_t::receive_port

<<**extension**>> (p. 795) 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. 2097)).

[**range**] [0,0xffffffff]

6.223 DDS_TransportMulticastSettingsSeq Struct Reference

Declares IDL sequence< **DDS_TransportMulticastSettings_t** (p. 2060) >

6.223.1 Detailed Description

Declares IDL sequence< **DDS_TransportMulticastSettings_t** (p. 2060) >

Instantiates:

<<**generic**>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_TransportMulticastSettings_t (p. 2060)

6.224 DDS_TransportPriorityQosPolicy Struct Reference

<<**spec**>> (p. 790) This QoS policy allows the application to take advantage of transports that are capable of sending messages with different priorities.

Data Fields

- **DDS_Long** value

<<**spec**>> (p. 790) This policy is a hint to the infrastructure as to how to set the priority of the underlying transport used to send the data.

6.224.1 Detailed Description

<<*spec*>> (p. 790) 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. 496). Also, many physical network transports themselves do not have a end-user controllable level of data packet priority.

Entity:

DDS_DataWriter (p. 485), **DDS_DataReader** (p. 615), **DDS_Topic** (p. 175)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

6.224.2 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 **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615). 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. 2140)), transport_priority_mapping_low (for example, see **NDDS_Transport_UDPv4_Property_t::transport_↵
priority_mapping_low** (p. 2140)), and transport_priority_mapping_high (for example, see **NDDS_Transport_UDPv4_↵
_Property_t::transport_priority_mapping_high** (p. 2140)). 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 **DDS_DataWriter** (p. 485) and a **DDS_DataReader** (p. 615). 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`.

6.224.3 Field Documentation

6.224.3.1 value

DDS_Long DDS_TransportPriorityQosPolicy::value

<<**spec**>> (p. 790) 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. 914), **Transports** (p. 908), **Concurrency** (p. 839)

6.225 DDS_TransportSelectionQosPolicy Struct Reference

<<**extension**>> (p. 795) Specifies the physical transports a **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615) may use to send or receive data.

Data Fields

- struct **DDS_StringSeq enabled_transports**

<<**extension**>> (p. 795) A sequence of transport aliases that specifies the transport instances available for use by the entity.

6.225.1 Detailed Description

<<**extension**>> (p. 795) Specifies the physical transports a **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615) 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 Connexx 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 **DDS_DataWriter** (p. 485) to send data or by a **DDS_DataReader** (p. 615) to receive data.

Entity:

DDS_DataReader (p. 615), **DDS_DataWriter** (p. 485)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

6.225.2 Field Documentation

6.225.2.1 enabled_transports

```
struct DDS_StringSeq DDS_TransportSelectionQosPolicy::enabled_transports
```

<<**extension**>> (p. 795) A sequence of transport aliases that specifies the transport instances available for use by the entity.

Of the transport instances installed with the **DDS_DomainParticipant** (p. 77), 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 **DDS_DomainParticipant** (p. 77).

An empty sequence is a special value that specifies all the transports installed with the **DDS_DomainParticipant** (p. 77).

The memory for the strings in this sequence is managed according to the conventions described in **String Conventions** (p. 1527). In particular, be careful to avoid a situation in which RTI Connexx allocates a string on your behalf and you then reuse that string in such a way that RTI Connexx 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. 1331). 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 **DDS_DomainParticipant** (p. 77).

[range] A sequence of non-null, non-empty strings.

See also

DDS_DomainParticipantQos::transport_builtin (p. 1729).

Categories

Immutable (p. 914), **Transports** (p. 908)

6.226 DDS_TransportUnicastQosPolicy Struct Reference

<<**extension**>> (p. 795) Specifies a subset of transports and a port number that can be used by an Entity to receive data.

Data Fields

- struct **DDS_TransportUnicastSettingsSeq** value
 <<**extension**>> (p. 795) A sequence of unicast communication settings.

6.226.1 Detailed Description

<<**extension**>> (p. 795) Specifies a subset of transports and a port number that can be used by an Entity to receive data.

Entity:

DDS_DomainParticipant (p. 77), **DDS_DataReader** (p. 615), **DDS_DataWriter** (p. 485)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

6.226.2 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 **DDS_Entity** (p. 1360) 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. 1901) associated with the **DDS_DomainParticipantQos** (p. 1726).

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. 2051) and **DDS_TransportSelection↔ QoSPolicy** (p. 2064)). Also, the associated ports are automatically determined (see **DDS_RtpsWellKnownPorts_t** (p. 1978)).

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 **DDS_DomainParticipant** (p. 77), this QoS policy sets the default list of addresses used by other applications to send user data for local DataReaders.
- For a **DDS_DataReader** (p. 615), 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 **DDS_DomainParticipant** (p. 77).
- For a reliable **DDS_DataWriter** (p. 485), 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 **DDS_DomainParticipant** (p. 77).

6.226.3 Field Documentation

6.226.3.1 value

```
struct DDS_TransportUnicastSettingsSeq DDS_TransportUnicastQosPolicy::value
```

<<**extension**>> (p. 795) A sequence of unicast communication settings.

An empty sequence means that applicable defaults specified by elsewhere (e.g. **DDS_DomainParticipantQos::default_unicast** (p. 1729)) 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. 1901) associated with the **DDS_DomainParticipantQos** (p. 1726).

[**default**] Empty sequence.

See also

DDS_DomainParticipantQos::default_unicast (p. 1729)

Categories

Immutable (p. 914), **Transports** (p. 908), **Discovery Metadata** (p. 958)

6.227 DDS_TransportUnicastSettings_t Struct Reference

Type representing a list of unicast locators.

Data Fields

- struct **DDS_StringSeq transports**
 <<**extension**>> (p. 795) 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. 795) The unicast port on which the entity can receive data.

6.227.1 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. 2065)

6.227.2 Field Documentation

6.227.2.1 transports

```
struct DDS_StringSeq DDS_TransportUnicastSettings_t::transports
```

<<**extension**>> (p. 795) 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. 1527). In particular, be careful to avoid a situation in which RTI Connexx allocates a string on your behalf and you then reuse that string in such a way that RTI Connexx 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. 1331).

[default] Empty sequence; i.e. all the transports available to the entity.

[range] Any sequence of non-null, non-empty strings.

6.227.2.2 receive_port

```
DDS_Long DDS_TransportUnicastSettings_t::receive_port
```

<<**extension**>> (p. 795) 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. 2097).

[range] [0,0xffffffff]

See also

DDS_WireProtocolQosPolicy::participant_id (p. 2097).

6.228 DDS_TransportUnicastSettingsSeq Struct Reference

Declares IDL `sequence< DDS_TransportUnicastSettings_t` (p. 2067) >

6.228.1 Detailed Description

Declares IDL `sequence< DDS_TransportUnicastSettings_t` (p. 2067) >

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_TransportUnicastSettings_t (p. 2067)

6.229 DDS_TrustAlgorithmRequirements Struct Reference

Type to describe Trust Plugins algorithm requirements for an entity.

Data Fields

- **DDS_TrustAlgorithmSet supported_mask**
Trust Plugins algorithms that an entity supports.
- **DDS_TrustAlgorithmSet required_mask**
Trust Plugins algorithms that an entity uses.

6.229.1 Detailed Description

Type to describe Trust Plugins algorithm requirements for an entity.

6.229.2 Field Documentation

6.229.2.1 supported_mask

DDS_TrustAlgorithmSet DDS_TrustAlgorithmRequirements::supported_mask

Trust Plugins algorithms that an entity supports.

6.229.2.2 required_mask

`DDS_TrustAlgorithmSet DDS_TrustAlgorithmRequirements::required_mask`

Trust Plugins algorithms that an entity uses.

6.230 DDS_TypeAllocationParams_t Struct Reference

Configures whether or not to allocate pointer and optional members.

Data Fields

- **DDS_Boolean allocate_pointers**
Whether to allocate pointer members or not.
- **DDS_Boolean allocate_optional_members**
Whether to allocate optional members or not.

6.230.1 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.c`, and `HelloWorldPlugin.c`.

6.230.2 Field Documentation

6.230.2.1 allocate_pointers

`DDS_Boolean DDS_TypeAllocationParams_t::allocate_pointers`

Whether to allocate pointer members or not.

Examples

`HelloWorld.c`, and `HelloWorldPlugin.c`.

6.230.2.2 allocate_optional_members

`DDS_Boolean DDS_TypeAllocationParams_t::allocate_optional_members`

Whether to allocate optional members or not.

6.231 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.

6.231.1 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. 2071) objects using the **DDS_TypeCodeFactory** (p. 2071) singleton. Then you can use the functions 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. 2071) object concurrently from multiple threads is *unsafe*. Modifying a **DDS_TypeCode** (p. 2071) from a single thread while concurrently reading the state of that **DDS_TypeCode** (p. 2071) from another thread is also *unsafe*. However, reading the state of a **DDS_TypeCode** (p. 2071) concurrently from multiple threads, without any modification, is *safe*.

Examples

HelloWorld.c, and **HelloWorldPlugin.c**.

6.232 DDS_TypeCodeFactory Struct Reference

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

6.232.1 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. 289) function.

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. 2071) class and then register it with the **Dynamic Data** (p. 313) API.

The functions 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. 291)

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. 298)
- **DDS_TypeCodeFactory_create_wstring_tc** (p. 298)
- **DDS_TypeCodeFactory_create_array_tc** (p. 300)
- **DDS_TypeCodeFactory_create_sequence_tc** (p. 299)

Creating definitions for structured types:

Structured types include structures, value types, and unions.

- **DDS_TypeCodeFactory_create_struct_tc** (p. 291)
- **DDS_TypeCodeFactory_create_value_tc** (p. 293)
- **DDS_TypeCodeFactory_create_union_tc** (p. 294)

Creating definitions for other types:

The type system also supports enumerations and aliases (i.e. `typedefs` in C and C++).

- **DDS_TypeCodeFactory_create_enum_tc** (p. 296)
- **DDS_TypeCodeFactory_create_alias_tc** (p. 297)

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. 2071) that you *created*; if you got the object from **DDS_TypeCodeFactory_get_primitive_tc** (p. 291), you must *not* delete it.)

- **DDS_TypeCodeFactory_delete_tc** (p. 290)

Copying type definitions:

You can also create deep copies of type definitions:

- **DDS_TypeCodeFactory_clone_tc** (p. 289)

6.233 DDS_TypeCodePrintFormatProperty Struct Reference

A collection of attributes used to configure how a TypeCode appears when converted to a string.

Data Fields

- **DDS_Long indent**
*Configures how much indent should be added to the string representation of a **DDS_TypeCode** (p. 2071).*
- **DDS_Boolean print_ordinals**
*Configures whether or not to print the ordinal value of each enumerator within a **DDS_TypeCode** (p. 2071).*
- **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.

6.233.1 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. 241).

6.233.2 Field Documentation

6.233.2.1 indent

DDS_Long DDS_TypeCodePrintFormatProperty::indent

Configures how much indent should be added to the string representation of a **DDS_TypeCode** (p. 2071).

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.

6.233.2.2 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. 2071).

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,  
};
```

6.233.2.3 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>
```

6.233.2.4 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;  
};
```

6.234 DDS_TypeConsistencyEnforcementQosPolicy Struct Reference

<<*spec*>> (p. 790) Defines the rules for determining whether the type used to publish a given topic is consistent with that used to subscribe to it.

Data Fields

- **DDS_TypeConsistencyKind kind**
 <<*spec*>> (p. 790) Type consistency kind.
- **DDS_Boolean ignore_sequence_bounds**
 <<*spec*>> (p. 790) Controls whether sequence bounds are taken into consideration for type assignability.
- **DDS_Boolean ignore_string_bounds**
 <<*spec*>> (p. 790) Controls whether string bounds are taken into consideration for type assignability.
- **DDS_Boolean ignore_member_names**
 <<*spec*>> (p. 790) Controls whether member names are taken into consideration for type assignability.
- **DDS_Boolean prevent_type_widening**
 <<*spec*>> (p. 790) Controls whether type widening is allowed.
- **DDS_Boolean force_type_validation**
 <<*spec*>> (p. 790) Controls whether type information must be available in order to complete matching between a **DDS_DataWriter** (p. 485) and a **DDS_DataReader** (p. 615).
- **DDS_Boolean ignore_enum_literal_names**
 <<*spec*>> (p. 790) Controls whether enumeration constant names are taken into consideration for type assignability.

6.234.1 Detailed Description

<<*spec*>> (p. 790) 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. 1341), **DDS_↔ALLOW_TYPE_COERCION** (p. 1341) and **DDS_AUTO_TYPE_COERCION** (p. 1341).

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. 1808) is updated.

The default enforcement kind is **DDS_AUTO_TYPE_COERCION** (p. 1341). This default kind translates to **DDS_↔ALLOW_TYPE_COERCION** (p. 1341) except in the following cases:

- When a **Zero Copy** (p. 210) DataReader is used, the kind is translated to **DDS_DISALLOW_TYPE_COERCION** (p. 1341).
- 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 TypeConsistencyEnforcement↔QosPolicy 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. 1341).

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:

DDS_DataReader (p. 615)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **UNTIL ENABLE** (p. 1245)

6.234.2 Field Documentation

6.234.2.1 kind

DDS_TypeConsistencyKind DDS_TypeConsistencyEnforcementQosPolicy::kind

<<*spec*>> (p. 790) Type consistency kind.

[default] **DDS_AUTO_TYPE_COERCION** (p. 1341)

Categories

Immutable (p. 914), **Matching** (p. 963), **Discovery Metadata** (p. 958), **Type System** (p. 912)

6.234.2.2 ignore_sequence_bounds

DDS_Boolean DDS_TypeConsistencyEnforcementQosPolicy::ignore_sequence_bounds

<<*spec*>> (p. 790) Controls whether sequence bounds are taken into consideration for type assignability.

If the option is set to **DDS_BOOLEAN_TRUE** (p. 1201), 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. 1201), then sequence bounds are taken into consideration for type assignability, and in order for T1 to be assignable from T2, it is required that $L1 \geq L2$.

[default] **DDS_BOOLEAN_TRUE** (p. 1201)

Categories

Immutable (p. 914), **Matching** (p. 963), **Discovery Metadata** (p. 958), **Type System** (p. 912)

6.234.2.3 ignore_string_bounds

DDS_Boolean DDS_TypeConsistencyEnforcementQosPolicy::ignore_string_bounds

<<*spec*>> (p. 790) Controls whether string bounds are taken into consideration for type assignability.

If the option is set to **DDS_BOOLEAN_TRUE** (p. 1201), 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. 1201), then string bounds are taken into consideration for type assignability, and in order for T1 to be assignable from T2, it is required that $L1 \geq L2$.

[default] **DDS_BOOLEAN_TRUE** (p. 1201)

Categories

Immutable (p. 914), **Matching** (p. 963), **Discovery Metadata** (p. 958), **Type System** (p. 912)

6.234.2.4 ignore_member_names

DDS_Boolean DDS_TypeConsistencyEnforcementQosPolicy::ignore_member_names

<<*spec*>> (p. 790) Controls whether member names are taken into consideration for type assignability.

If the option is set to **DDS_BOOLEAN_TRUE** (p. 1201), then member names are not considered as part of the type assignability. If the option is set to **DDS_BOOLEAN_FALSE** (p. 1201), 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. 1201)

Categories

Immutable (p. 914), **Matching** (p. 963), **Discovery Metadata** (p. 958), **Type System** (p. 912)

6.234.2.5 prevent_type_widening

DDS_Boolean DDS_TypeConsistencyEnforcementQosPolicy::prevent_type_widening

<<*spec*>> (p. 790) Controls whether type widening is allowed.

If the option is set to **DDS_BOOLEAN_FALSE** (p. 1201), then type widening is permitted. If the option is set to **DDS_BOOLEAN_TRUE** (p. 1201), then a wider type may not be assignable from a narrower type.

[default] **DDS_BOOLEAN_FALSE** (p. 1201)

Categories

Immutable (p. 914), **Matching** (p. 963), **Discovery Metadata** (p. 958), **Type System** (p. 912)

6.234.2.6 force_type_validation

DDS_Boolean DDS_TypeConsistencyEnforcementQosPolicy::force_type_validation

<<*spec*>> (p. 790) Controls whether type information must be available in order to complete matching between a **DDS_DataWriter** (p. 485) and a **DDS_DataReader** (p. 615).

If the option is set to **DDS_BOOLEAN_TRUE** (p. 1201), then type information must be available in order to complete matching between a **DDS_DataWriter** (p. 485) and a **DDS_DataReader** (p. 615). If the option is set to **DDS_BOOLEAN_FALSE** (p. 1201), 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. 1201)

Categories

Immutable (p. 914), **Matching** (p. 963), **Discovery Metadata** (p. 958), **Type System** (p. 912)

6.234.2.7 ignore_enum_literal_names

DDS_Boolean DDS_TypeConsistencyEnforcementQosPolicy::ignore_enum_literal_names

<<*spec*>> (p. 790) Controls whether enumeration constant names are taken into consideration for type assignability.

If the option is set to **DDS_BOOLEAN_TRUE** (p. 1201), 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. 1201), 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. 1201)

Categories

Immutable (p. 914), **Matching** (p. 963), **Discovery Metadata** (p. 958), **Type System** (p. 912)

6.235 DDS_TypeDeallocationParams_t Struct Reference

Configures whether to release or not pointer and optional members.

Data Fields

- **DDS_Boolean delete_pointers**
Whether to delete pointer members or not.
- **DDS_Boolean delete_optional_members**
Whether to delete optional members or not.

6.235.1 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.c, and **HelloWorldPlugin.c**.

6.235.2 Field Documentation

6.235.2.1 delete_pointers

```
DDS_Boolean DDS_TypeDeallocationParams_t::delete_pointers
```

Whether to delete pointer members or not.

Examples

HelloWorld.c, and **HelloWorldPlugin.c**.

6.235.2.2 delete_optional_members

```
DDS_Boolean DDS_TypeDeallocationParams_t::delete_optional_members
```

Whether to delete optional members or not.

Examples

HelloWorld.c, and **HelloWorldPlugin.c**.

6.236 DDS_TypeSupportQosPolicy Struct Reference

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Allows you to attach application-specific values to a **DDS_Data**↔**Writer** (p. 485) or **DDS_DataReader** (p. 615), 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.

Data Fields

- void * **plugin_data**
[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Value to pass into the type plugin's de-/serialization function
- **DDS_CdrPaddingKind** **cdr_padding_kind**
[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Determines whether or not the padding bytes will be set to zero during CDR serialization.

6.236.1 Detailed Description

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Allows you to attach application-specific values to a **DDS_Data**↔**Writer** (p. 485) or **DDS_DataReader** (p. 615), 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:

DDS_DomainParticipant (p. 77), **DDS_DataReader** (p. 615), **DDS_DataWriter** (p. 485)

Properties:

RxO (p. 1245) = NO

Changeable (p. 1245) = **UNTIL ENABLE** (p. 1245)

6.236.2 Usage

The `DDS_TypeSupportQosPolicy::plugin_data` (p. 2082) allows you to associate a pointer to an object with a `DDS_↵_DataWriter` (p. 485) or `DDS_DataReader` (p. 615). This object pointer is passed to the serialization routine of the data type associated with the `DDS_DataWriter` (p. 485) or the deserialization routine of the data type associated with the `DDS_DataReader` (p. 615).

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 `DDS_DataWriter` (p. 485) or received by a `DDS_Data↵Reader` (p. 615) 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 `DDS_DataWriter` (p. 485) calls `FooDataWriter_write` (p. 496), and then never again.

The `DDS_TypeSupportQosPolicy::cdr_padding_kind` (p. 2082) allows you to choose whether or not the padding bytes are set to zero during CDR serialization.

6.236.3 Field Documentation

6.236.3.1 plugin_data

```
void* DDS_TypeSupportQosPolicy::plugin_data
```

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) Value to pass into the type plugin's de-/serialization function

[default] NULL

Categories

Immutable (p. 914), **Application-Defined Metadata** (p. 836)

6.236.3.2 cdr_padding_kind

`DDS_CdrPaddingKind DDS_TypeSupportQosPolicy::cdr_padding_kind`

[DEPRECATED] (p. 965) <<*extension*>> (p. 795) 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. 1342) defaults to **DDS_NOT_SET_CDR_PADDING** (p. 1342).

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. 1342) means that the entity will inherit whatever value is set on the DomainParticipant.

[default] **DDS_AUTO_CDR_PADDING** (p. 1342)

Categories

Immutable (p. 914), **Security** (p. 906)

6.237 DDS_UInt8Seq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_UInt8** (p. 1202) >

6.237.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_UInt8** (p. 1202) >

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_UInt8 (p. 1202)

FooSeq (p. 2113)

6.238 DDS_UnionMember Struct Reference

A description of a member of a union.

Data Fields

- `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.

6.238.1 Detailed Description

A description of a member of a union.

See also

`DDS_UnionMemberSeq` (p. 2085)

`DDS_TypeCodeFactory_create_union_tc` (p. 294)

6.238.2 Field Documentation

6.238.2.1 name

```
char* DDS_UnionMember::name
```

The name of the union member.

Cannot be NULL.

6.238.2.2 is_pointer

```
DDS_Boolean DDS_UnionMember::is_pointer
```

Indicates whether the union member is a pointer or not.

6.238.2.3 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. 1203) the label value should be evaluated to an integer value. For instance, 'a' would be evaluated to 97.

6.238.2.4 type

```
const DDS_TypeCode* DDS_UnionMember::type
```

The type of the union member.

Cannot be NULL.

6.239 DDS_UnionMemberSeq Struct Reference

Defines a sequence of union members.

6.239.1 Detailed Description

Defines a sequence of union members.

See also

DDS_UnionMember (p. 2083)

FooSeq (p. 2113)

DDS_TypeCodeFactory_create_union_tc (p. 294)

6.240 DDS_UnsignedLongLongSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_UnsignedLongLong** (p. 1203) >

6.240.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_UnsignedLongLong** (p. 1203) >

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_UnsignedLongLong (p. 1203)

FooSeq (p. 2113)

6.241 DDS_UnsignedLongSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_UnsignedLong** (p. 1203) >

6.241.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_UnsignedLong** (p. 1203) >

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_UnsignedLong (p. 1203)

FooSeq (p. 2113)

6.242 DDS_UnsignedShortSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_UnsignedShort** (p. 1202) >

6.242.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_UnsignedShort** (p. 1202) >

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_UnsignedShort (p. 1202)

FooSeq (p. 2113)

6.243 DDS_UserDataQosPolicy Struct Reference

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 165) during discovery.

Data Fields

- struct **DDS_OctetSeq** value
 <<*spec*>> (p. 790) a sequence of octets

6.243.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 165) during discovery.

Entity:

DDS_DomainParticipant (p. 77), **DDS_DataReader** (p. 615), **DDS_DataWriter** (p. 485)

Properties:

RxO (p. 1245) = NO;
Changeable (p. 1245) = YES (p. 1245)

See also

DDS_DomainParticipant_get_builtin_subscriber (p. 129)

6.243.2 Usage

The purpose of this QoS is to allow the application to attach additional information to the created **DDS_Entity** (p. 1360) 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 Connex.

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 **DDS_DomainParticipant_ignore_participant** (p. 131), **DDS_DomainParticipant_ignore_publication** (p. 132), **DDS_DomainParticipant_ignore_subscription** (p. 133), and **DDS_DomainParticipant_ignore_topic** (p. 132), 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 Connex stores the data placed in this policy in pre-allocated pools. It is therefore necessary to configure RTI Connex 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. 1746), **DDS_DomainParticipantResourceLimitsQosPolicy::writer_user_data_max_length** (p. 1747), and **DDS_DomainParticipantResourceLimitsQosPolicy::reader_user_data_max_length** (p. 1747).

6.243.3 Field Documentation

6.243.3.1 value

```
struct DDS_OctetSeq DDS_UserDataQosPolicy::value
```

<<*spec*>> (p. 790) a sequence of octets

[**default**] empty (zero-length)

[**range**] Octet sequence of length [0,max_length]

Categories

Mutable (p. 949), **Application-Defined Metadata** (p. 836), **Discovery Metadata** (p. 958)

6.244 DDS_ValueMember Struct Reference

A description of a member of a value type.

Data Fields

- 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.

6.244.1 Detailed Description

A description of a member of a value type.

See also

DDS_ValueMemberSeq (p. 2090)

DDS_TypeCodeFactory_create_value_tc (p. 293)

6.244.2 Field Documentation

6.244.2.1 name

```
char* DDS_ValueMember::name
```

The name of the value member.

Cannot be NULL.

6.244.2.2 type

```
const DDS_TypeCode* DDS_ValueMember::type
```

The type of the value member.

Cannot be NULL.

6.244.2.3 is_pointer

```
DDS_Boolean DDS_ValueMember::is_pointer
```

Indicates whether the value member is a pointer or not.

6.244.2.4 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. 238).

6.244.2.5 is_key

```
DDS_Boolean DDS_ValueMember::is_key
```

Indicates if the value member is a key member or not.

6.244.2.6 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. 239) or **DDS_PUBLIC_MEMBER** (p. 240).

6.244.2.7 id

`DDS_Long DDS_ValueMember::id`

The member ID.

Use **DDS_TYPECODE_MEMBER_ID_INVALID** (p. 237) to have the member ID automatically assigned.

6.244.2.8 is_optional

`DDS_Boolean DDS_ValueMember::is_optional`

Indicates if the value member is optional or required.

6.245 DDS_ValueMemberSeq Struct Reference

Defines a sequence of value members.

6.245.1 Detailed Description

Defines a sequence of value members.

See also

DDS_ValueMember (p. 2088)

FooSeq (p. 2113)

DDS_TypeCodeFactory_create_value_tc (p. 293)

6.246 DDS_VendorId_t Struct Reference

<<extension>> (p. 1012) Type used to represent the vendor of the service implementing the RTPS protocol.

Data Fields

- **DDS_Octet vendorId** [DDS_VENDOR_ID_LENGTH_MAX]
The vendor Id.

6.246.1 Detailed Description

<<**extension**>> (p. 1012) Type used to represent the vendor of the service implementing the RTPS protocol.

6.246.2 Field Documentation

6.246.2.1 vendorId

```
DDS_Octet DDS_VendorId_t::vendorId[ DDS_VENDOR_ID_LENGTH_MAX]
```

The vendor Id.

6.247 DDS_VirtualWriterInfo Struct Reference

The structure that contains the information of a virtual writer.

6.247.1 Detailed Description

The structure that contains the information of a virtual writer.

6.248 DDS_WaitSetProperty_t Struct Reference

<<**extension**>> (p. 1012) Specifies the **DDS_WaitSet** (p. 1371) behavior for multiple trigger events.

Data Fields

- **DDS_Long max_event_count**
*Maximum number of trigger events to cause a **DDS_WaitSet** (p. 1371) to awaken.*
- struct **DDS_Duration_t max_event_delay**
*Maximum delay from occurrence of first trigger event to cause a **DDS_WaitSet** (p. 1371) to awaken.*

6.248.1 Detailed Description

<<*extension*>> (p. 1012) Specifies the **DDS_WaitSet** (p. 1371) behavior for multiple trigger events.

In simple use, a **DDS_WaitSet** (p. 1371) returns when a single trigger event occurs on one of its attached **DDS_↔Condition** (p. 1370) (s), or when the `timeout` maximum wait duration specified in the **DDS_WaitSet_wait** (p. 1382) call expires.

The **DDS_WaitSetProperty_t** (p. 2091) allows configuration of the waiting behavior of a **DDS_WaitSet** (p. 1371). 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 **DDS_WaitSet_wait** (p. 1382) call continues to apply.

Entity:

DDS_WaitSet (p. 1371)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **YES** (p. 1245)

6.248.2 Field Documentation

6.248.2.1 `max_event_count`

DDS_Long `DDS_WaitSetProperty_t::max_event_count`

Maximum number of trigger events to cause a **DDS_WaitSet** (p. 1371) to awaken.

The **DDS_WaitSet** (p. 1371) will wait until up to `max_event_count` trigger events have occurred before returning. The **DDS_WaitSet** (p. 1371) 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

;

6.248.2.2 max_event_delay

```
struct DDS_Duration_t DDS_WaitSetProperty_t::max_event_delay
```

Maximum delay from occurrence of first trigger event to cause a **DDS_WaitSet** (p. 1371) to awaken.

The **DDS_WaitSet** (p. 1371) 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 **DDS_WaitSet** (p. 1371).

Note that **DDS_RETCODE_TIMEOUT** (p. 1222) is *not* returned if `max_event_delay` is exceeded. **DDS_RETCODE_TIMEOUT** (p. 1222) is returned only if the `timeout` duration expires before any trigger events occur.

[default] **DDS_DURATION_INFINITE** (p. 1208) ;

6.249 DDS_WcharSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_Wchar** (p. 1202) >

6.249.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_Wchar** (p. 1202) >

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_Wchar (p. 1202)

FooSeq (p. 2113)

6.250 DDS_WireProtocolQosPolicy Struct Reference

<<*extension*>> (p. 795) Specifies the wire-protocol-related attributes for the **DDS_DomainParticipant** (p. 77).

Data Fields

- **DDS_Long participant_id**
 <<*basic*>> (p. 783) <<*extension*>> (p. 795) A value used to distinguish among different participants belonging to the same domain on the same host.
- **DDS_UnsignedLong rtps_host_id**
 <<*extension*>> (p. 795) The RTPS Host ID of the domain participant.
- **DDS_UnsignedLong rtps_app_id**
 <<*extension*>> (p. 795) The RTPS App ID of the domain participant.
- **DDS_UnsignedLong rtps_instance_id**
 <<*extension*>> (p. 795) The RTPS Instance ID of the **DDS_DomainParticipant** (p. 77).
- struct **DDS_RtpsWellKnownPorts_t rtps_well_known_ports**
 <<*extension*>> (p. 795) Configures the RTPS well-known port mappings.
- **DDS_RtpsReservedPortKindMask rtps_reserved_port_mask**
 <<*extension*>> (p. 795) Specifies which well-known ports to reserve when enabling the participant.
- **DDS_WireProtocolQosPolicyAutoKind rtps_auto_id_kind**
 <<*extension*>> (p. 795) Kind of auto mechanism used to calculate the GUID prefix.
- **DDS_Boolean compute_crc**
 <<*extension*>> (p. 795) Adds RTPS CRC submessage to every message when this field is set to **DDS_BOOLEAN_TRUE** (p. 1201).
- **DDS_Boolean check_crc**
 <<*extension*>> (p. 795) 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. 1201).

6.250.1 Detailed Description

<<*extension*>> (p. 795) Specifies the wire-protocol-related attributes for the **DDS_DomainParticipant** (p. 77).

Entity:

DDS_DomainParticipant (p. 77)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = **NO** (p. 1245)

6.250.2 Usage

This QoS policy configures some participant-wide properties of the DDS Real-Time Publish Subscribe (RTPS) on-the-wire protocol. (**DDS_DataWriterProtocolQosPolicy** (p. 1649) and **DDS_DataReaderProtocolQosPolicy** (p. 1596) configure RTPS and reliability properties on a per **DDS_DataWriter** (p. 485) or **DDS_DataReader** (p. 615) 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 **DDS_DomainParticipant** (p. 77) 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 **DDS_DomainParticipant** (p. 77) 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. 2100) 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. 2097) (first integer), **DDS_WireProtocolQosPolicy::rtps_app_id** (p. 2098) (second integer), and **DDS_WireProtocolQosPolicy::rtps_instance_id** (p. 2098) (third integer).

The **DDS_WireProtocolQosPolicy::rtps_auto_id_kind** (p. 2100) field supports three different prefix generation algorithms:

1. In the default and most common scenario, **DDS_WireProtocolQosPolicy::rtps_auto_id_kind** (p. 2100) is set to **DDS_RTPS_AUTO_ID_FROM_UUID** (p. 1348). 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. 1348), 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. 1348), 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. 1348).

First, get the participant QoS from the DomainParticipantFactory:

```
DDS_DomainParticipantFactory_get_default_participant_qos(
    DDS_DomainParticipantFactory_get_instance(),
    &participant_qos);
```

Second, change the **DDS_WireProtocolQoSPolicy** (p. 2093) using one of the options shown below.

Third, create the **DDS_DomainParticipant** (p. 77) as usual, using the modified QoS structure instead of the default one.

Option 1: Use **DDS_RTPS_AUTO_ID_FROM_MAC** (p. 1348) 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. 1348) 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 Connexthandle 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. 1348) 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 Connexthandle will take responsibility for them). RTI recommends that you always specify the two parts separately in order to clearly show the difference.

6.250.3 Field Documentation

6.250.3.1 participant_id

DDS_Long DDS_WireProtocolQosPolicy::participant_id

<<**basic**>> (p. 783) <<**extension**>> (p. 795) 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. 1616) or **DDS_DataWriterQos::unicast** (p. 1671)).

For more information on port mapping, please refer to **DDS_RtpsWellKnownPorts_t** (p. 1978).

Each **DDS_DomainParticipant** (p. 77) 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 Connexthandle will *automatically* resolve the participant ID as follows.

- RTI Connexthandle will pick the *smallest* participant ID based on the unicast ports available on the transports enabled for discovery.
- RTI Connexthandle will attempt to resolve an automatic port index either when a DomainParticipant 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 DomainParticipant is enabled.

[default] -1 [automatic], i.e. RTI Connexthandle will automatically pick the **participant_id**, as described above.

[range] [≥ 0], or -1, and does not violate guidelines stated in **DDS_RtpsWellKnownPorts_t** (p. 1978).

See also

DDS_Entity_enable() (p. 1363)

NDDS_Transport_Support_register_transport() (p. 730)

Categories

Immutable (p. 914), **Discovery** (p. 853)

6.250.3.2 rtps_host_id

DDS_UnsignedLong DDS_WireProtocolQosPolicy::rtps_host_id

<<**extension**>> (p. 795) 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. 2100) is equal to DDS_RTPS_AUTO_ID_FROM_IP (p. 1348), 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. 2100) is equal to DDS_RTPS_AUTO_ID_FROM_MAC (p. 1348), 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. 2100) is equal to DDS_RTPS_AUTO_ID_FROM_UUID (p. 1348), the value will be the first 32 bits of the GUID prefix assigned by the UUID algorithm.

[range] [0,0xffffffff]

Categories

Immutable (p. 914), **Discovery** (p. 853)

6.250.3.3 rtps_app_id

DDS_UnsignedLong DDS_WireProtocolQosPolicy::rtps_app_id

<<**extension**>> (p. 795) 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. 2100) is equal to DDS_RTPS_AUTO_ID_FROM_IP (p. 1348) the value will be the process (or task) ID.

If DDS_WireProtocolQosPolicy::rtps_auto_id_kind (p. 2100) is equal to DDS_RTPS_AUTO_ID_FROM_MAC (p. 1348) 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. 2100) is equal to DDS_RTPS_AUTO_ID_FROM_UUID (p. 1348), the value will be the middle 32 bits of the GUID prefix assigned by the UUID algorithm.

[range] [0,0xffffffff]

Categories

Immutable (p. 914), **Discovery** (p. 853)

6.250.3.4 rtps_instance_id

`DDS_UnsignedLong DDS_WireProtocolQosPolicy::rtps_instance_id`

<<*extension*>> (p. 795) The RTPS Instance ID of the **DDS_DomainParticipant** (p. 77).

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. 2100) is equal to **DDS_RTPS_AUTO_ID_FROM_IP** (p. 1348), 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. 2100) is equal to **DDS_RTPS_AUTO_ID_FROM_MAC** (p. 1348), 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 participant.

If **DDS_WireProtocolQosPolicy::rtps_auto_id_kind** (p. 2100) is equal to **DDS_RTPS_AUTO_ID_FROM_UUID** (p. 1348), 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. 1348) 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. 914), **Discovery** (p. 853)

6.250.3.5 rtps_well_known_ports

`struct DDS_RtpsWellKnownPorts_t DDS_WireProtocolQosPolicy::rtps_well_known_ports`

<<*extension*>> (p. 795) 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. 1349)

Categories

Immutable (p. 914), **Discovery** (p. 853)

6.250.3.6 rtps_reserved_port_mask

DDS_RtpsReservedPortKindMask `DDS_WireProtocolQosPolicy::rtps_reserved_port_mask`

<<**extension**>> (p. 795) 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. 1978) will be detected at this time, and the enable operation will fail.

[default] **DDS_RTPS_RESERVED_PORT_MASK_DEFAULT** (p. 1345)

Categories

Immutable (p. 914), **Discovery** (p. 853)

6.250.3.7 rtps_auto_id_kind

DDS_WireProtocolQosPolicyAutoKind `DDS_WireProtocolQosPolicy::rtps_auto_id_kind`

<<**extension**>> (p. 795) Kind of auto mechanism used to calculate the GUID prefix.

[default] **DDS_RTPS_AUTO_ID_FROM_UUID** (p. 1348)

Categories

Immutable (p. 914), **Discovery** (p. 853)

6.250.3.8 compute_crc

DDS_Boolean `DDS_WireProtocolQosPolicy::compute_crc`

<<**extension**>> (p. 795) Adds RTPS CRC submessage to every message when this field is set to **DDS_BOOLEAN_↔_TRUE** (p. 1201).

The computed CRC covers the entire RTPS message excluding the RTPS header.

[default] **DDS_BOOLEAN_FALSE** (p. 1201)

Categories

Immutable (p. 914), **Security** (p. 906)

6.250.3.9 check_crc

DDS_Boolean DDS_WireProtocolQosPolicy::check_crc

<<**extension**>> (p. 795) 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. 1201).

DDS_WireProtocolQosPolicy::compute_crc (p. 2100) must be enabled at the publishing application for validating the message at the subscribing application.

[default] **DDS_BOOLEAN_FALSE** (p. 1201)

Categories

Immutable (p. 914), **Security** (p. 906)

6.251 DDS_WriteParams_t Struct Reference

<<**extension**>> (p. 1012) Input parameters for writing with **FooDataWriter_write_w_params** (p. 501), **FooDataWriter_dispose_w_params** (p. 504), **FooDataWriter_register_instance_w_params** (p. 493), **FooDataWriter_unregister_instance_w_params** (p. 496)

Data Fields

- **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.

6.251.1 Detailed Description

<<*extension*>> (p. 1012) Input parameters for writing with **FooDataWriter_write_w_params** (p. 501), **FooDataWriter_dispose_w_params** (p. 504), **FooDataWriter_register_instance_w_params** (p. 493), **FooDataWriter_unregister_instance_w_params** (p. 496)

6.251.2 Field Documentation

6.251.2.1 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. 1391)) receive their actual value after **FooDataWriter_write_w_params** (p. 501) is called.

To reset those fields to their automatic value after calling **FooDataWriter_write_w_params** (p. 501), use **DDS_WriteParams_reset** (p. 1391)

6.251.2.2 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. 1650).
- 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. 1993) and **DDS_SampleInfo::original_publication_virtual_sequence_number** (p. 1993) in the **DDS_SampleInfo** (p. 1985).

[default] **DDS_AUTO_SAMPLE_IDENTITY** (p. 1391).

6.251.2.3 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. 1985) (see **DDS_SampleInfo_get_related_sample_identity** (p. 702)).

The default value is **DDS_UNKNOWN_SAMPLE_IDENTITY** (p. 1392), 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. 1993) and **DDS_SampleInfo::related_original_publication_virtual_↵sequence_number** (p. 1994) in the **DDS_SampleInfo** (p. 1985).

[default] **DDS_UNKNOWN_SAMPLE_IDENTITY** (p. 1392)

6.251.2.4 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 **DDS_DataReader** (p. 615) objects by means of the `source_timestamp` attribute within the **DDS_SampleInfo** (p. 1985).

[default] **DDS_TIME_INVALID** (p. 1208).

6.251.2.5 cookie

```
struct DDS_Cookie_t DDS_WriteParams_t::cookie
```

Octet sequence identifying written data sample.

Used in the callback **DDS_DataWriterListener::on_sample_removed** (p. 1647) to associate a removed sample with a written sample.

[default] Empty sequence (zero-length).

6.251.2.6 handle

```
DDS_InstanceHandle_t DDS_WriteParams_t::handle
```

Instance handle.

Either the handle returned by a previous call to **FooDataWriter_register_instance** (p. 491), or else the special value **DDS_HANDLE_NIL** (p. 228).

[default] **DDS_HANDLE_NIL** (p. 228)

6.251.2.7 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. 1319)) with **DDS_FlowControllerProperty_t::scheduling_policy** (p. 1799) set to **DDS_HPF_FLOW** ↔ **CONTROLLER_SCHED_POLICY** (p. 561).

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. 1318), 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. 1318), 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. 2101) of **FooDataWriter_write_w_params** (p. 501).

For dispose and unregister samples, use the **DDS_WriteParams_t** (p. 2101) of **FooDataWriter_dispose_w_params** (p. 504) and **FooDataWriter_unregister_instance_w_params** (p. 496).

[default] 0 (lowest priority)

See also

DDS_ChannelSettings_t::priority (p. 1565)

6.251.2.8 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. 1388)) is reserved for marking samples as redelivered when using RTI Queuing Service.

The second bit (**DDS_INTERMEDIATE_REPLY_SEQUENCE_SAMPLE** (p. 1388)) 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. 1994).

[default] 0 (no flags are set)

6.251.2.9 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. 1985) (see **DDS_SampleInfo::source_guid** (p. 1994)).

The default value is **DDS_GUID_AUTO** (p. 1213), 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. 2105) 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. 1388)) 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. 1213) (the `source_guid` is automatically set to the **DDS_DataWriter** (p. 485) virtual GUID).

See also

DDS_WriteParams_t::related_source_guid (p. 2105)

6.251.2.10 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. 1985) (see **DDS_SampleInfo::related_source_guid** (p. 1994)).

This field and **DDS_WriteParams_t::source_guid** (p. 2104) 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. 1213), and is not propagated.

[default] **DDS_GUID_UNKNOWN** (p. 1213)

See also

DDS_WriteParams_t::source_guid (p. 2104)

6.251.2.11 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. 1985) (see `DDS_SampleInfo::related_subscription_guid` (p. 1995)).

The default value is `DDS_GUID_UNKNOWN` (p. 1213), 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. 501) function 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. 1213)

6.252 DDS_WriterDataLifecycleQosPolicy Struct Reference

<<*spec*>> (p. 790) Controls how a `DDS_DataWriter` (p. 485) handles the lifecycle of the instances (keys) that it is registered to manage.

Data Fields

- **DDS_Boolean autodispose_unregistered_instances**
<<*spec*>> (p. 790) Boolean flag that controls whether or not a writer will also dispose of an instance when the `DDS_DataWriter` (p. 485) unregisters from the instance by means of the unregister operations.
- struct **DDS_Duration_t autopurge_unregistered_instances_delay**
<<*extension*>> (p. 795) Maximum duration for which the `DDS_DataWriter` (p. 485) will maintain information regarding an instance once it has unregistered the instance.
- struct **DDS_Duration_t autopurge_disposed_instances_delay**
<<*extension*>> (p. 795) Maximum duration for which the `DDS_DataWriter` (p. 485) will maintain information regarding an instance once it has disposed the instance.

6.252.1 Detailed Description

<<*spec*>> (p. 790) Controls how a `DDS_DataWriter` (p. 485) handles the lifecycle of the instances (keys) that it is registered to manage.

Entity:

DDS_DataWriter (p. 485)

Properties:

RxO (p. 1245) = N/A

Changeable (p. 1245) = YES (p. 1245)

6.252.2 Usage

This policy determines how the **DDS_DataWriter** (p. 485) acts with regards to the lifecycle of the data instances it manages (data instances that have been either explicitly registered with the **DDS_DataWriter** (p. 485) or implicitly registered by directly writing the data).

You may use **FooDataWriter_unregister_instance** (p. 493) to indicate that the **DDS_DataWriter** (p. 485) no longer wants to send data for a **DDS_Topic** (p. 175).

The behavior controlled by this QoS policy applies on a per instance (key) basis for keyed Topics, so that when a **DDS_↔_DataWriter** (p. 485) 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. 1863)), 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.

6.252.3 Field Documentation

6.252.3.1 autodispose_unregistered_instances

DDS_Boolean DDS_WriterDataLifecycleQosPolicy::autodispose_unregistered_instances

<<*spec*>> (p. 790) Boolean flag that controls whether or not a writer will also dispose of an instance when the **DDS_↔_DataWriter** (p. 485) unregisters from the instance by means of the unregister operations.

- **DDS_BOOLEAN_TRUE** (p. 1201)
The **DDS_DataWriter** (p. 485) 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. 1201) (default)
The **DDS_DataWriter** (p. 485) 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. 1201)

Categories

Immutable (p. 914), **Instance Lifecycle** (p. 867), **Sample Memory Management** (p. 902)

6.252.3.2 autopurge_unregistered_instances_delay

```
struct DDS_Duration_t DDS_WriterDataLifecycleQosPolicy::autopurge_unregistered_instances_delay
```

<<**extension**>> (p. 795) Maximum duration for which the **DDS_DataWriter** (p. 485) will maintain information regarding an instance once it has unregistered the instance.

Determines how long the **DDS_DataWriter** (p. 485) will maintain information regarding an instance that has been unregistered. By default, the **DDS_DataWriter** (p. 485) 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. 1951) is exceeded. This behavior can be changed by setting `autopurge_unregistered_instances_delay` to a value other than **DDS_DURATION_INFINITE** (p. 1208).

After this time elapses, the **DDS_DataWriter** (p. 485) will purge all internal information regarding the instance, including historical samples, even if **DDS_ResourceLimitsQosPolicy::max_instances** (p. 1951) 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. 1208) value.

[default] **DDS_DURATION_INFINITE** (p. 1208) (disabled) for all **DDS_DataWriter** (p. 485) except for the built-in discovery DataWriters

DDS_DURATION_ZERO (p. 1209) for built-in discovery DataWriters (see **DDS_DiscoveryConfigQosPolicy::publication_writer_data_lifecycle** (p. 1699), **DDS_DiscoveryConfigQosPolicy::subscription_writer_data_lifecycle** (p. 1700) and **DDS_DiscoveryConfigQosPolicy::participant_configuration_writer_data_lifecycle** (p. 1713)).

[range] [0, 1 year] or **DDS_DURATION_INFINITE** (p. 1208)

Categories

Immutable (p. 914), **Instance Lifecycle** (p. 867), **Sample Memory Management** (p. 902)

6.252.3.3 autopurge_disposed_instances_delay

```
struct DDS_Duration_t DDS_WriterDataLifecycleQosPolicy::autopurge_disposed_instances_delay
```

<<**extension**>> (p. 795) Maximum duration for which the **DDS_DataWriter** (p. 485) will maintain information regarding an instance once it has disposed the instance.

Determines how long the **DDS_DataWriter** (p. 485) 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. 1208), 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. 1209) and **DDS_DURATION_INFINITE** (p. 1208) values (finite values are not supported).

[default] **DDS_DURATION_INFINITE** (p. 1208) (disabled)

[range] [0, 1 year] or **DDS_DURATION_INFINITE** (p. 1208)

Categories

Immutable (p. 914), **Instance Lifecycle** (p. 867), **Sample Memory Management** (p. 902)

6.253 DDS_WstringSeq Struct Reference

Instantiates **FooSeq** (p. 2113) < **DDS_Wchar** (p. 1202)* >

6.253.1 Detailed Description

Instantiates **FooSeq** (p. 2113) < **DDS_Wchar** (p. 1202)* >

Instantiates:

<<*generic*>> (p. 1013) **FooSeq** (p. 2113)

See also

DDS_Wchar (p. 1202)

DDS_StringSeq (p. 2005)

FooSeq (p. 2113)

6.254 Foo Struct Reference

A representative user-defined data type.

6.254.1 Detailed Description

A representative user-defined data type.

Foo (p. 2109) represents a user-defined data-type that is intended to be distributed using DDS.

The type **Foo** (p. 2109) 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. 2113) 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. 2112) and **FooDataWriter** (p. 2113) types that allow the application to send and receive data of this type.

See also

FooSeq (p. 2113), **FooDataWriter** (p. 2113), **FooDataReader** (p. 2112), **FooTypeSupport** (p. 2114), the `Code Generator User's Manual`

6.255 FooBarReplier Struct Reference

Allows receiving requests and sending replies.

6.255.1 Detailed Description

Allows receiving requests and sending replies.

A Replier is an entity with two associated **topics** (p. 167): 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. 305), and **DDS_DynamicData** (p. 1768). **Note:** At this moment, in the C version of this API, only rtiddsgen-generated types are supported.

To create a Replier for two types, a request type TReq=Foo (p. 2109) and a reply type TRep=Bar, your application needs to instantiate the data structure **FooBarReplier** (p. 2110) and the specific operations that can publish and subscribe to those types. In this documentation we refer to the type-dependent operations as **FooBarReplier_** (for example, **FooBarReplier_take_requests** (p. 771)). Some operations are type-independent and their names always begin with **RTI_Connext_Replier_** (for example, **RTI_Connext_Replier_wait_for_requests** (p. 768)). See **Creating a Replier** (p. 1007).

A Replier 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 original Requester can identify it)

For multi-reply scenarios in which a **FooBarReplier** (p. 2110) generates more than one reply for a request, the **FooBarReplier** (p. 2110) should mark all the intermediate replies (all but the last) using the **DDS_INTERMEDIATE_REPLY_SEQUENCE_SAMPLE** (p. 1388) flag in **DDS_WriteParams_t::flag** (p. 2104).

Much like a Requester, a Replier has an associated **DDS_DomainParticipant** (p. 77), which can be shared with other Repliers or RTI Connext routines. All the other entities required for the request-reply interaction, including a **DDS_DataWriter** (p. 485) for writing replies and a **DDS_DataReader** (p. 615) for reading requests, are automatically created when the Replier is constructed.

Quality of Service for the underlying DataWriter and DataReader can be configured (see **RTI_Connext_Requester_Params::qos_profile_name** (p. 2175)). By default, they are created with **DDS_RELIABLE_RELIABILITY_QOS** (p. 1323). The exact default configuration is described here: **Configuring Request-Reply QoS profiles** (p. 1010)

There are several ways to use a Replier:

- A thread **receives** (p. 770) requests and then dispatches them. If the computation of a reply is a simple operation, consider using a **FooBarSimpleReplier** (p. 2112) instead of a Replier.
- Polling without waiting, using **FooBarReplier_take_requests** (p. 771) directly.
- Using a **RTI_Connext_ReplierListener** (p. 2168) to get notified and **get** (p. 771) the requests within the callback.

See also

FooBarRequester (p. 2111)

Request-Reply Examples (p. 1002)

Replier example (p. 1008)

6.256 FooBarRequester Struct Reference

Allows sending requests and receiving replies.

6.256.1 Detailed Description

Allows sending requests and receiving replies.

A requester is an entity with two associated **topics** (p. 167): 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. 305), and **DDS_DynamicData** (p. 1768). **Note:** At this moment, in the C version of this API, only `rtiddsgen`-generated types are supported.

To create a Requester for two types, a request type `TReq=Foo` (p. 2109) and a reply type `TRep=Bar`, your application needs to instantiate the data structure **FooBarRequester** (p. 2111) and the specific operations that can publish and subscribe to those types. In this documentation we refer to the type-dependent operations as `FooBarRequester_` (for example, **FooBarRequester_take_replies** (p. 756)). Some operations are type-independent and their name always begins with `RTI_Connext_Requester_` (for example, **RTI_Connext_Requester_wait_for_replies** (p. 750)).

See **Creating a Requester** (p. 1004) to see how to instantiate a **FooBarRequester** (p. 2111).

A Replier and a Requester communicate when they use the same topics for requests and replies (see **RTI_Connext_RequesterParams::service_name** (p. 2174)) on the same **domain** (p. 77).

A Requester 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 receives multiple replies from a Replier for a given request, the Requester 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. 1388) is set in **DDS_SampleInfo::flag** (p. 1994) after receiving each reply. This indicates it is NOT the last reply.

A requester has an associated **DDS_DomainParticipant** (p. 77), 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 **DDS_DataWriter** (p. 485) for writing requests and a **DDS_DataReader** (p. 615) for reading replies, are automatically created when the requester is constructed.

Quality of Service for the underlying `DataWriter` and `DataReader` can be configured (see **RTI_Connext_RequesterParams::qos_profile_name** (p. 2175)). By default, they are created with **DDS_RELIABLE_RELIABILITY_QOS** (p. 1323). The exact default configuration is described here: **Configuring Request-Reply QoS profiles** (p. 1010)

See also

FooBarReplier (p. 2110)

Request-Reply Examples (p. 1002)

Requester example (p. 1005)

6.257 FooBarSimpleReplier Struct Reference

A callback-based replier.

6.257.1 Detailed Description

A callback-based replier.

A SimpleReplier is based on a **RTI_Connext_SimpleReplierListener** (p. 2176) that users provide . Requests are passed to the callback, which returns a reply. The reply is directed only to the Requester 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 **FooBarReplier** (p. 2110) instead.

See also

FooBarReplier (p. 2110)

RTI_Connext_SimpleReplierListener (p. 2176)

SimpleReplier example (p. 1009)

6.258 FooDataReader Struct Reference

<<*interface*>> (p. 1013) <<*generic*>> (p. 1013) User data type-specific data reader.

6.258.1 Detailed Description

<<*interface*>> (p. 1013) <<*generic*>> (p. 1013) 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

DDS_DataReader (p. 615)

Foo (p. 2109)

FooDataWriter (p. 2113)

the Code Generator User's Manual

6.259 FooDataWriter Struct Reference

<<*interface*>> (p. 1013) <<*generic*>> (p. 1013) User data type specific data writer.

6.259.1 Detailed Description

<<*interface*>> (p. 1013) <<*generic*>> (p. 1013) 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

DDS_DataWriter (p. 485)

Foo (p. 2109)

FooDataReader (p. 2112)

the Code Generator User's Manual

6.260 FooSeq Struct Reference

<<*interface*>> (p. 1013) <<*generic*>> (p. 1013) A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as **Foo** (p. 2109).

6.260.1 Detailed Description

<<*interface*>> (p. 1013) <<*generic*>> (p. 1013) A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as **Foo** (p. 2109).

For users who define data types in OMG IDL, this type corresponds to the IDL express `sequence<Foo (p. 2109)>`.

For any user-data type **Foo** (p. 2109) that an application defines for the purpose of data-distribution with RTI Connext, a **FooSeq** (p. 2113) is generated. We refer to an IDL `sequence<Foo (p. 2109)>` as **FooSeq** (p. 2113).

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_get_maximum** (p. 1514) operation.
- The 'length' represents the actual number of elements it currently holds. It is returned by the **FooSeq_get_length** (p. 1515) operation.

- The 'owned' flag represents whether the sequence owns the underlying buffer. It is returned by the **FooSeq_↔
has_ownership** (p. 1524) 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. 1201), 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. 1201), 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. 1201)

See also

FooDataWriter (p.2113), **FooDataReader** (p.2112), **FooTypeSupport** (p.2114), the `Code Generator User's Manual`

6.261 FooTypeSupport Struct Reference

<<*interface*>> (p. 1013) <<*generic*>> (p. 1013) User data type specific interface.

6.261.1 Detailed Description

<<*interface*>> (p. 1013) <<*generic*>> (p. 1013) 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_C (p. 213)
the `Code Generator User's Manual`

6.262 NDDS_Config_LibraryVersion_t Struct Reference

The version of a single library shipped as part of an RTI Connext distribution.

Data Fields

- **DDS_Long major**
The major version of a single RTI Connex library.
- **DDS_Long minor**
The minor version of a single RTI Connex library.
- char **release**
The release letter of a single RTI Connex library.
- **DDS_Long build**
The build number of a single RTI Connex library.

6.262.1 Detailed Description

The version of a single library shipped as part of an RTI Connex distribution.

RTI Connex is comprised of a number of separate libraries. Although RTI Connex 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.

6.262.2 Field Documentation

6.262.2.1 major

DDS_Long NDDS_Config_LibraryVersion_t::major

The major version of a single RTI Connex library.

6.262.2.2 minor

DDS_Long NDDS_Config_LibraryVersion_t::minor

The minor version of a single RTI Connex library.

6.262.2.3 release

char NDDS_Config_LibraryVersion_t::release

The release letter of a single RTI Connex library.

6.262.2.4 build

`DDS_Long NDDS_Config_LibraryVersion_t::build`

The build number of a single RTI Connex library.

6.263 NDDS_Config_Logger Struct Reference

<<*interface*>> (p. 1013) The singleton type used to configure RTI Connex logging.

6.263.1 Detailed Description

<<*interface*>> (p. 1013) The singleton type used to configure RTI Connex logging.

6.264 NDDS_Config_LoggerDevice Struct Reference

<<*interface*>> (p. 1013) Logging device interface. Use for user-defined logging devices.

Data Fields

- **NDDS_Config_LoggerDeviceWriteFnc write**
Write a log message to a specified logging device.
- **NDDS_Config_LoggerDeviceCloseFnc close**
Close the logging device.

6.264.1 Detailed Description

<<*interface*>> (p. 1013) 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 Connex to the standard output.

You can use the function **NDDS_Config_Logger_set_output_file** (p. 1450) to redirect the log messages to a file.

To further customize the management of generated log messages, the logger offers the function **NDDS_Config_Logger_set_output_device** (p. 1457) 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 Connex core library, including calls to **DDS_DomainParticipant_get_current_time** (p. 138), from any of the logging device operations.

6.264.2 Field Documentation

6.264.2.1 write

NDDS_Config_LoggerDeviceWriteFnc NDDS_Config_LoggerDevice::write

Write a log message to a specified logging device.

Note: It is not safe to make any calls to the RTI Connex core library, including calls to **DDS_DomainParticipant_get_↵_current_time** (p. 138), from any of the logging device operations.

6.264.2.2 close

NDDS_Config_LoggerDeviceCloseFnc NDDS_Config_LoggerDevice::close

Close the logging device.

Note: It is not safe to make any calls to the RTI Connex core library, including calls to **DDS_DomainParticipant_get_↵_current_time** (p. 138), from any of the logging device operations.

6.265 NDDS_Config_LogMessage Struct Reference

A log message's input parameter to a logging device, including the text and additional information.

Data Fields

- **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. 1446).*

6.265.1 Detailed Description

A log message's input parameter to a logging device, including the text and additional information.

6.265.2 Field Documentation

6.265.2.1 text

```
const char* NDDS_Config_LogMessage::text
```

Message text.

6.265.2.2 level

```
NDDS_Config_LogLevel NDDS_Config_LogMessage::level
```

Message level.

6.265.2.3 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).

6.265.2.4 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.

6.265.2.5 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 DataWriterQos" and "ERROR: Failed to get TopicName". In this case, the message_id is associated to the get failure.

6.265.2.6 timestamp

```
struct DDS_Duration_t NDDS_Config_LogMessage::timestamp
```

The time when the log message was printed.

6.265.2.7 facility

```
NDDS_Config_LogFacility NDDS_Config_LogMessage::facility
```

The Facility associated with the log message. See **NDDS_Config_LogFacility** (p. 1446).

6.266 NDDS_Config_Version_t Struct Reference

<<*interface*>> (p. 1013) The version of an RTI Connext distribution.

6.266.1 Detailed Description

<<*interface*>> (p. 1013) 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.

6.267 NDDS_Transport_Address_t Struct Reference

Addresses are stored individually as network-ordered bytes.

```
#include <transport_common_user.h>
```

Data Fields

- unsigned char **network_ordered_value** [NDDS_TRANSPORT_ADDRESS_LENGTH]

6.267.1 Detailed Description

Addresses are stored individually as network-ordered bytes.

RTI Connex addresses are numerically stored in a transport independent manner. RTI Connex uses a IPv6-compatible format, which means that the data structure to hold an **NDDS_Transport_Address_t** (p.2120) 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 Connex 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.2120) 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]`).

6.267.2 Field Documentation

6.267.2.1 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).

6.268 NDDS_Transport_Interface_t Struct Reference

Storage for the description of a network interface used by a Transport Plugin.

```
#include <transport_common.h>
```


Data Fields

- **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.

6.268.1 Detailed Description

Storage for the description of a network interface used by a Transport Plugin.

6.268.2 Field Documentation

6.268.2.1 transport_classid

NDDS_Transport_ClassId_t NDDS_Transport_Interface_t::transport_classid

The transport classid of the interface.

6.268.2.2 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.

6.268.2.3 status

NDDS_Transport_Interface_Status_t NDDS_Transport_Interface_t::status

The state of the interface.

6.268.2.4 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.

6.269 NDDS_Transport_Property_t Struct Reference

Base configuration structure that must be inherited by derived Transport Plugin classes.

```
#include <transport_interface_user.h>
```

Data Fields

- **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 Connex core.
- RTI_INT32 **gather_send_buffer_count_max**
Specifies the maximum number of buffers that RTI Connex can pass to the send() function 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.

6.269.1 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 Connex. The RTI Connex 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. 2122) structure.

For example,

```
struct MyTransportPluginProperty_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.

6.269.2 Field Documentation

6.269.2.1 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. 1025) 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.

6.269.2.2 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. 722)

6.269.2.3 properties_bitmap

```
RTI_INT32 NDDS_Transport_Property_t::properties_bitmap
```

A bitmap that defines various properties of the transport to the RTI Connex core.

Currently, the only property supported is whether or not the transport plugin will always loan a buffer when RTI Connex 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. 1025)

6.269.2.4 gather_send_buffer_count_max

```
RTI_INT32 NDDS_Transport_Property_t::gather_send_buffer_count_max
```

Specifies the maximum number of buffers that RTI Connex can pass to the `send()` function of a transport plugin.

The transport plugin `send()` API supports a gather-send concept, where the `send()` call can take several discontinuous buffers, assemble and send them in a single message. This enables RTI Connex 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 Connex from trying to gather too many buffers into a send call for the transport plugin.

RTI Connex 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. 1026).

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 Connex layer.

6.269.2.5 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`.

6.269.2.6 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 DomainParticipant. 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 **DDS_DomainParticipant** (p. 77) is deleted.

[default] empty list that represents all available interfaces.

See also

NDDS_Transport_Property_t::max_interface_count (p. 2128)

6.269.2.7 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 Connex 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

6.269.2.8 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 DomainParticipant. 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 Connex 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 **DDS_DomainParticipant** (p. 77) is deleted.

[default] empty list that represents no denied interfaces.

6.269.2.9 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 Connex 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

6.269.2.10 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 **DDS_DomainParticipant** (p. 77) is deleted.

[default] empty list that represents all available interfaces.

6.269.2.11 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

6.269.2.12 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 **DDS_DomainParticipant** (p. 77) is deleted.

[default] empty list that represents no denied interfaces.

6.269.2.13 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

6.269.2.14 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.

6.269.2.15 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 (default value), then the participant identifier is computed as the last 5 digits of the rtps_instance_id in the participant GUID.

6.269.2.16 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_interface_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. 2125)

6.270 NDDS_Transport_Shmem_Property_t Struct Reference

Subclass of **NDDS_Transport_Property_t** (p. 2122) allowing specification of parameters that are specific to the shared-memory transport.

```
#include <transport_shmem.h>
```

Data Fields

- 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.
- RTI_Bool **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. 2131) 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. 2131) is set to '1'.*

6.270.1 Detailed Description

Subclass of **NDDS_Transport_Property_t** (p. 2122) allowing specification of parameters that are specific to the shared-memory transport.

See also

NDDS_Transport_Support_set_builtin_transport_property() (p. 735)

6.270.2 Field Documentation

6.270.2.1 parent

```
struct NDDS_Transport_Property_t NDDS_Transport_Shmem_Property_t::parent
```

Generic properties of all transport plugins.

6.270.2.2 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. 2124). 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. 2122), **NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT** (p. 1040)

6.270.2.3 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. 1040)

6.270.2.4 enable_udp_debugging

RTI_Bool 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. 2131) :: **NDDS_Transport_Shmem_Property_t::udp_debugging_port** (p. 2132).

[default] 0

6.270.2.5 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. 2131) is set to '1'.

[default] 239.255.1.2

6.270.2.6 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. 2131) is set to '1'.

[default] 7399

6.271 NDDS_Transport_Support Struct Reference

<<*interface*>> (p. 1013) The utility class used to configure RTI Connexx pluggable transports.

6.271.1 Detailed Description

<<*interface*>> (p. 1013) The utility class used to configure RTI Connexx pluggable transports.

6.272 NDDS_Transport_UDP_WAN_CommPortsMappingInfo Struct Reference

Type for storing UDP WAN communication ports.

```
#include <transport_udp.h>
```

Data Fields

- **NDDS_Transport_Port_t** rtps_port
RTPS port.
- **NDDS_Transport_UDP_Port** host_port
Host port.
- **NDDS_Transport_UDP_Port** public_port
Public port.

6.272.1 Detailed Description

Type for storing UDP WAN communication ports.

6.272.2 Field Documentation

6.272.2.1 rtps_port

NDDS_Transport_Port_t NDDS_Transport_UDP_WAN_CommPortsMappingInfo::rtps_port

RTPS port.

6.272.2.2 host_port

NDDS_Transport_UDP_Port NDDS_Transport_UDP_WAN_CommPortsMappingInfo::host_port

Host port.

6.272.2.3 public_port

NDDS_Transport_UDP_Port NDDS_Transport_UDP_WAN_CommPortsMappingInfo::public_port

Public port.

6.273 NDDS_Transport_UDPv4_Property_t Struct Reference

Configurable IPv4/UDP Transport-Plugin properties.

```
#include <transport_udpv4.h>
```

Data Fields

- 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 **DDS_DomainParticipant** (p. 77).*

6.273.1 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

NDDS_Transport_Support_set_builtin_transport_property() (p. 735)

NDDS_Transport_UDPv4_new (p. 1049)

6.273.2 Field Documentation

6.273.2.1 parent

```
struct NDDS_Transport_Property_t NDDS_Transport_UDPv4_Property_t::parent
```

Generic properties of all transport plugins.

6.273.2.2 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. 2124). 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. 1047).

The maximum value for this property depends on the operating system.

Valid values are:

- **NDDS_TRANSPORT_UDPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT** (p.1047): `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.1047): `setsockopt()` (or equivalent) will not be called to size the send buffer of the socket. RTI Connex will use the OS default.
- **NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX** (p.1046): RTI Connex 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.

6.273.2.3 `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 `RCVBUF` to the value of this parameter.

This value must be greater than or equal to **NDDS_Transport_Property_t::message_size_max** (p.2124). 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.1047).

The maximum value for this property depends on the operating system.

Valid values are:

- **NDDS_TRANSPORT_UDPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT** (p.1047): `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.1047): `setsockopt()` (or equivalent) will not be called to size the receive buffer of the socket. RTI Connex will use the OS default.
- **NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX** (p.1046): RTI Connex 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.

6.273.2.4 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.

6.273.2.5 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.

6.273.2.6 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. 1048)

6.273.2.7 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 Connex 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.]

6.273.2.8 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 Connexx decide between the above two choices.

The current "automatic" (-1) RTI Connexx 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 Connexx policy based on availability of the shared memory transport.

6.273.2.9 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.2142) 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

6.273.2.10 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)

6.273.2.11 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 Connexx will use the zero-copy API if offered by the OS.

6.273.2.12 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

6.273.2.13 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 `DDS_DomainParticipant_get_participant_protocol_status` (p. 128).

[default] 1 (enabled)

6.273.2.14 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. 2140) and `NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_high` (p. 2140) to define the mapping from the DDS transport priority (see `TRANSPORT_PRIORITY` (p. 1337)) 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.

6.273.2.15 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. 2140) and `NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_high` (p. 2140) 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.

6.273.2.16 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.2140) and **NDDS_Transport_UDPv4_Property_t::transport_priority_mapping_low** (p.2140) 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.

6.273.2.17 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)

6.273.2.18 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).

6.273.2.19 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.

6.273.2.20 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.

6.273.2.21 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.2124) 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.2124)

6.273.2.22 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 Connex 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 Connex 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

6.273.2.23 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

6.273.2.24 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 **DDS_DomainParticipant** (p. 77) 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. 1978)). 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 **DDS_DomainParticipant** (p. 77) 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)

6.273.2.25 port_offset

```
RTI_INT32 NDDS_Transport_UDPv4_Property_t::port_offset
```

Port offset to allow multiple instances of UDPv4 transports in the same **DDS_DomainParticipant** (p. 77).

By default, it is not possible to have multiple instances of the UDPv4 transport in the same **DDS_DomainParticipant** (p. 77). 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 **DDS_DomainParticipant** (p. 77).

An RTPS port will be mapped to an UDP port as follows: UDP port = RTPS port + port_offset.

[default] 0

6.274 NDDS_Transport_UDPv4_WAN_Property_t Struct Reference

Configurable IPv4/UDP WAN Transport-Plugin properties.

```
#include <transport_udpv4.h>
```

Data Fields

- 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. 2151).
- 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.

6.274.1 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

`NDDS_Transport_Support_set_builtin_transport_property()` (p. 735)

`NDDS_Transport_UDPv4_WAN_new` (p. 1056)

6.274.2 Field Documentation

6.274.2.1 parent

```
struct NDDS_Transport_Property_t NDDS_Transport_UDPv4_WAN_Property_t::parent
```

Generic properties of all transport plugins.

6.274.2.2 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. 2135)

6.274.2.3 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. 2136)

6.274.2.4 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 Connexx decide between the above two choices.

The current "automatic" (-1) RTI Connexx 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 Connexx policy based on availability of the shared memory transport.

6.274.2.5 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.2150) 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

6.274.2.6 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)

6.274.2.7 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 Connexant will use the zero-copy API if offered by the OS.

6.274.2.8 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.

6.274.2.9 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 **DDS_DomainParticipant_get_participant_protocol_status** (p. 128).

[default] 1 (enabled)

6.274.2.10 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.2149) and **NDDS_Transport_UDPv4_WAN_Property_t::transport_priority_mapping_high** (p.2149) to define the mapping from the DDS transport priority (see **TRANSPORT_PRIORITY** (p.1337)) 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.

6.274.2.11 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.2148) and **NDDS_Transport_UDPv4_WAN_Property_t::transport_priority_mapping_high** (p.2149) 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.

6.274.2.12 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.2148) and **NDDS_Transport_UDPv4_WAN_Property_t::transport_priority_mapping_low** (p.2149) 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.

6.274.2.13 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)

6.274.2.14 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).

6.274.2.15 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.

6.274.2.16 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.2124) 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.2124)

6.274.2.17 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 Connex 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 Connex 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

6.274.2.18 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 **DDS_↔_DomainParticipant** (p. 77) (publicly reachable) in order to support the two communication scenarios shown in the Figure below.

For an external **DDS_DomainParticipant** (p. 77) 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)

6.274.2.19 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>
  <udp4_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>
  </udp4_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)).

6.274.2.20 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. 2151).

[default] 0

6.274.2.21 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

6.274.2.22 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)

6.275 NDDS_Transport_UDPv6_Property_t Struct Reference

Configurable IPv6/UDP Transport-Plugin properties.

```
#include <transport_udp6.h>
```

Data Fields

- 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 **DDS_DomainParticipant** (p. 77).*

6.275.1 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

NDDS_Transport_Support_set_builtin_transport_property() (p. 735)

NDDS_Transport_UDPv6_new (p. 1065)

6.275.2 Field Documentation

6.275.2.1 parent

```
struct NDDS_Transport_Property_t NDDS_Transport_UDPv6_Property_t::parent
```

Generic properties of all transport plugins.

6.275.2.2 `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.2124). 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. 1063).

The maximum value for this property depends on the operating system.

Valid values are:

- `NDDS_TRANSPORT_UDPv6_SEND_SOCKET_BUFFER_SIZE_DEFAULT` (p.1063): `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. 1063): `setsockopt()` (or equivalent) will not be called to size the send buffer of the socket. RTI Connex will use the OS default.
- `NDDS_TRANSPORT_UDPv6_SOCKET_BUFFER_SIZE_OS_MAX` (p. 1062): RTI Connex 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.

6.275.2.3 `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 `RCVBUF` to the value of this parameter.

This value must be greater than or equal to `NDDS_Transport_Property_t::message_size_max` (p.2124). 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. 1063).

The maximum value for this property depends on the operating system.

Valid values are:

- `NDDS_TRANSPORT_UDPv6_RECV_SOCKET_BUFFER_SIZE_DEFAULT` (p.1063): `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. 1063): `setsockopt()` (or equivalent) will not be called to size the receive buffer of the socket. RTI Connex will use the OS default.
- `NDDS_TRANSPORT_UDPv6_SOCKET_BUFFER_SIZE_OS_MAX` (p. 1062): RTI Connex 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.

6.275.2.4 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.

6.275.2.5 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.

6.275.2.6 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. 1064)

6.275.2.7 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.

6.275.2.8 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 Connexx decide between the above two choices.

The current "automatic" (-1) RTI Connexx 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 Connexx policy based on availability of the shared memory transport.

6.275.2.9 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)

6.275.2.10 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 Connexx will use the zero copy API if offered by the OS.

6.275.2.11 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.

6.275.2.12 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.

6.275.2.13 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.2160) and **NDDS_Transport_UDPv6_Property_t::transport_priority_mapping_high** (p.2160) to define the mapping from the DDS transport priority (see **TRANSPORT_PRIORITY** (p.1337)) 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.

6.275.2.14 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.2159) and **NDDS_Transport_UDPv6_Property_t::transport_priority_mapping_high** (p.2160) 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.

6.275.2.15 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.2159) and **NDDS_Transport_UDPv6_Property_t::transport_priority_mapping_low** (p.2160) 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.

6.275.2.16 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. 2141)

6.275.2.17 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. 2141)

6.275.2.18 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. 2141)

6.275.2.19 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. 2141)

6.275.2.20 `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.2124) 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.2124)

6.275.2.21 `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.2142)

6.275.2.22 `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.2142)

6.275.2.23 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 **DDS_DomainParticipant** (p. 77) 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. 1978)). 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 **DDS_DomainParticipant** (p. 77) 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)

6.275.2.24 port_offset

```
RTI_INT32 NDDS_Transport_UDPv6_Property_t::port_offset
```

Port offset to allow multiple instances of UDPv6 transports in the same **DDS_DomainParticipant** (p. 77).

By default, it is not possible to have multiple instances of the UDPv6 transport in the same **DDS_DomainParticipant** (p. 77). 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 **DDS_DomainParticipant** (p. 77).

An RTPS port will be mapped to an UDP port as follows: UDP port = RTPS port + port_offset.

[default] 0

6.276 NDDS_Transport_UUID Struct Reference

Unequivocally identifies a transport plugin instance.

```
#include <transport_common.h>
```

6.276.1 Detailed Description

Unequivocally identifies a transport plugin instance.

6.277 NDDS_Utility_HeapMonitoringParams_t Struct Reference

Input parameters for enabling heap monitoring. They will be used for configuring the format of the snapshot.

Data Fields

- **NDDS_Utility_HeapMonitoringSnapshotOutputFormat** `snapshot_output_format`
Specify the format of the output of the snapshot.
- **NDDS_Utility_HeapMonitoringSnapshotContentFormat** `snapshot_content_format`
It is a bitmap to decide which information of the snapshot will be displayed.

6.277.1 Detailed Description

Input parameters for enabling heap monitoring. They will be used for configuring the format of the snapshot.

6.277.2 Field Documentation

6.277.2.1 `snapshot_output_format`

NDDS_Utility_HeapMonitoringSnapshotOutputFormat `NDDS_Utility_HeapMonitoringParams_t::snapshot_↔
output_format`

Specify the format of the output of the snapshot.

The enum can take two options:

- **NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_OUTPUT_FORMAT_STANDARD** (p. 1479):
The output of the snapshot will be in plain text.
- **NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_OUTPUT_FORMAT_COMPRESSED** (p. 1479):
The output of the snapshot will be compressed using Zlib technology.
The file can be uncompressed using zlib-flate.

[default] **NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_OUTPUT_FORMAT_STANDARD** (p. 1479).

6.277.2.2 snapshot_content_format

NDDS_Utility_HeapMonitoringSnapshotContentFormat NDDS_Utility_HeapMonitoringParams_t::snapshot_content_format

It is a bitmap to decide which information of the snapshot will be displayed.

``suggestion Bitmap that specifies the field to display in the snapshot.

You can combine these values by logically ORing them together. For example: (NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT_TOPIC (p. 1479) | NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT_FUNCTION (p. 1479))

[default] NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_DEFAULT (p. 1479)

6.278 NDDS_Utility_NetworkCaptureParams_t Struct Reference

Input parameters for starting network capture.

Data Fields

- 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).

6.278.1 Detailed Description

Input parameters for starting network capture.

6.278.2 Field Documentation

6.278.2.1 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.

6.278.2.2 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. 1488) and **NDDS_Utility_NetworkCaptureContentKindMask** (p. 1488) .

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. 1486).

6.278.2.3 traffic

```
NDDS_Utility_NetworkCaptureTrafficKindMask NDDS_Utility_NetworkCaptureParams_t::traffic
```

Traffic direction to capture.

It accepts values from **NDDS_Utility_NetworkCaptureTrafficKind** (p. 1489) and **NDDS_Utility_NetworkCaptureTrafficKindMask** (p. 1488) .

[default] Capture both inbound and outbound traffic - **NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_DEFAULT** (p. 1487).

6.278.2.4 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. 1201)

6.278.2.5 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. 2029). Default value of (**DDS_THREAD_SETTINGS_PRIORITY_ENFORCE** (p. 1237) | **DDS_THREAD_SETTINGS_STDIO** (p. 1237)).
- **DDS_ThreadSettings_t::priority** (p. 2030). Platform-dependent - Consult `Platform Notes` for additional details.
- **DDS_ThreadSettings_t::stack_size** (p. 2030). Platform-dependent - Consult `Platform Notes` for additional details.

6.278.2.6 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).

6.279 RTI_Connext_Replier Struct Reference

The type-independent version of a Replier.

6.279.1 Detailed Description

The type-independent version of a Replier.

Any `FooBarReplier*` can be safely cast to `RTI_Connext_Replier*`.

See also

FooBarReplier (p. 2110)

FooBarSimpleReplier (p. 2112)

6.280 RTI_Connext_ReplierListener Struct Reference

Called when a **FooBarReplier** (p. 2110) has new available requests.

Data Fields

- **RTI_Connext_ReplierListener_OnRequestAvailableCallback on_request_available**
User callback.
- `void *` **user_data**
User data attached to a listener.

6.280.1 Detailed Description

Called when a **FooBarReplier** (p. 2110) has new available requests.

A repplier listener is a way to implement a callback that will be invoked when requests are available. It is an optional **RTI_Connext_ReplierParams** (p. 2169).

This listener simply notifies when requests are available. The callback implementation can then use **FooBarReplier_↔take_requests** (p. 771) 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 **FooBarSimpleReplier** (p. 2112).

See also

FooBarReplier_create_w_params (p. 769)

6.280.2 Field Documentation

6.280.2.1 user_data

```
void* RTI_Connext_ReplierListener::user_data
```

User data attached to a listener.

6.281 RTI_Connext_ReplierParams Struct Reference

Contains the parameters for creating a **FooBarReplier** (p. 2110).

Data Fields

- **DDS_DomainParticipant * participant**
The participant this replier uses to join a domain.
- char * **service_name**
The service name the Replier offers and Requesters use to match.
- char * **request_topic_name**
Sets a specific request topic name.
- char * **reply_topic_name**
Sets a specific reply topic name.
- char * **qos_library_name**
*Sets the library that contains the **RTI_Connext_ReplierParams::qos_profile_name** (p. 2171).*
- char * **qos_profile_name**
Sets a QoS profile for the entities in this replier.
- const struct **DDS_DataWriterQos * datawriter_qos**
Sets the quality of service of the reply DataWriter.
- const struct **DDS_DataReaderQos * datareader_qos**
Sets the quality of service of the request DataReader.
- **DDS_Publisher * publisher**
Sets a specific Publisher.
- **DDS_Subscriber * subscriber**
Sets a specific Subscriber.
- struct **RTI_Connext_ReplierListener * listener**
Sets a listener that is called when requests are available.

6.281.1 Detailed Description

Contains the parameters for creating a **FooBarReplier** (p. 2110).

See also

RTI_Connext_RequesterParams (p. 2173)

6.281.2 Field Documentation

6.281.2.1 participant

```
DDS_DomainParticipant* RTI_Connext_ReplierParams::participant
```

The participant this replier uses to join a domain.

6.281.2.2 service_name

```
char* RTI_Connext_ReplierParams::service_name
```

The service name the Replier offers and Requesters use to match.

See also

RTI_Connext_RequesterParams::service_name (p.2174)

6.281.2.3 request_topic_name

```
char* RTI_Connext_ReplierParams::request_topic_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 **RTI_Connext_ReplierParams::service_name** (p.2170). If that topic already exists, it will be reused. This is useful to have the Replier use a **DDS_ContentFilteredTopic** (p.177) to receive only certain requests.

6.281.2.4 reply_topic_name

```
char* RTI_Connext_ReplierParams::reply_topic_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 c the **RTI_Connext_ReplierParams::service_name** (p.2170). If that topic already exists, it will be reused.

6.281.2.5 qos_library_name

```
char* RTI_Connext_ReplierParams::qos_library_name
```

Sets the library that contains the **RTI_Connext_ReplierParams::qos_profile_name** (p. 2171).

See also

RTI_Connext_ReplierParams::qos_profile_name (p. 2171)

6.281.2.6 qos_profile_name

```
char* RTI_Connext_ReplierParams::qos_profile_name
```

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's underlying reply DataWriter and request DataReader.

Parameters

<i>qos_library_name</i>	The name of the QoS library
<i>qos_profile_name</i>	The name of the QoS profile inside the QoS library

See also

RTI_Connext_RequesterParams::qos_profile_name (p. 2175)

Configuring Request-Reply QoS profiles (p. 1010)

Configuring QoS Profiles with XML (p. 971)

6.281.2.7 datawriter_qos

```
const struct DDS_DataWriterQos* RTI_Connext_ReplierParams::datawriter_qos
```

Sets the quality of service of the reply DataWriter.

See also

RTI_Connext_ReplierParams::qos_profile_name (p. 2171)

6.281.2.8 datareader_qos

```
const struct DDS_DataReaderQos* RTI_Connext_ReplierParams::datareader_qos
```

Sets the quality of service of the request DataReader.

See also

RTI_Connext_ReplierParams::qos_profile_name (p. 2171)

6.281.2.9 publisher

```
DDS_Publisher* RTI_Connext_ReplierParams::publisher
```

Sets a specific Publisher.

See also

RTI_Connext_RequesterParams::publisher (p. 2176)

6.281.2.10 subscriber

```
DDS_Subscriber* RTI_Connext_ReplierParams::subscriber
```

Sets a specific Subscriber.

See also

RTI_Connext_RequesterParams::subscriber (p. 2176)

6.281.2.11 listener

```
struct RTI_Connext_ReplierListener* RTI_Connext_ReplierParams::listener
```

Sets a listener that is called when requests are available.

See also

RTI_Connext_ReplierListener (p. 2168)

6.282 RTI_Connext_Requester Struct Reference

The type-independent version of a Requester.

6.282.1 Detailed Description

The type-independent version of a Requester.

Any `FooBarRequester*` can be safely cast to `RTI_Connext_Requester*`.

See also

FooBarRequester (p. 2111)

6.283 RTI_Connext_RequesterParams Struct Reference

Contains the parameters for creating a **FooBarRequester** (p. 2111).

Data Fields

- **DDS_DomainParticipant * participant**
The participant this requester uses to join a DDS domain.
- **char * service_name**
The service name that Repliers and Requester use to match and communicate.
- **char * request_topic_name**
Sets a specific request topic name.
- **char * reply_topic_name**
Sets a specific reply topic name.
- **char * qos_library_name**
Sets the library that contains the `RTI_Connext_RequesterParams::qos_profile_name` (p. 2175).
- **char * qos_profile_name**
Sets a QoS profile for the entities in this requester.
- **const struct DDS_DataWriterQos * datawriter_qos**
Sets the quality of service of the request DataWriter.
- **const struct DDS_DataReaderQos * datareader_qos**
Sets the quality of service of the request DataReader.
- **DDS_Publisher * publisher**
Sets a specific Publisher.
- **DDS_Subscriber * subscriber**
Sets a specific Subscriber.

6.283.1 Detailed Description

Contains the parameters for creating a **FooBarRequester** (p. 2111).

The following parameters are required to create a Requester:

- A **DDS_DomainParticipant** (p. 77) (**RTI_Connext_RequesterParams::participant** (p. 2174)), and
- Either a service name (**RTI_Connext_RequesterParams::service_name** (p. 2174))
- Or custom topic names (**RTI_Connext_RequesterParams::request_topic_name** (p. 2174) and **RTI_Connext_RequesterParams::reply_topic_name** (p. 2175))

The rest of the parameters that can be set in a RequesterParams object are optional.

See also

Creating a Requester with optional parameters (p. 1004)

6.283.2 Field Documentation

6.283.2.1 participant

```
DDS_DomainParticipant* RTI_Connext_RequesterParams::participant
```

The participant this requester uses to join a DDS domain.

6.283.2.2 service_name

```
char* RTI_Connext_RequesterParams::service_name
```

The service name that Repliers and Requester use to match and communicate.

A Requester and a Replier 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 and Replier 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 **RTI_Connext_RequesterParams::request_topic_name** (p. 2174) and **RTI_Connext_RequesterParams::reply_topic_name** (p. 2175).

6.283.2.3 request_topic_name

```
char* RTI_Connext_RequesterParams::request_topic_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 **RTI_Connext_RequesterParams::service_name** (p. 2174). If that topic already exists, it will be reused.

6.283.2.4 reply_topic_name

```
char* RTI_Connext_RequesterParams::reply_topic_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 **RTI_Connext_RequesterParams::service_name** (p. 2174). If that topic already exists, it will be reused.

6.283.2.5 qos_library_name

```
char* RTI_Connext_RequesterParams::qos_library_name
```

Sets the library that contains the **RTI_Connext_RequesterParams::qos_profile_name** (p. 2175).

See also

RTI_Connext_RequesterParams::qos_profile_name (p. 2175)

6.283.2.6 qos_profile_name

```
char* RTI_Connext_RequesterParams::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'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 (**RTI_Connext_RequesterParams::datawriter_qos** (p. 2175), **RTI_Connext_RequesterParams::datareader_qos** (p. 2176)).

The library where this profile is located has to be set with **RTI_Connext_RequesterParams::qos_library_name** (p. 2175)

See also

RTI_Connext_RequesterParams::qos_library_name (p. 2175)

Configuring Request-Reply QoS profiles (p. 1010)

Configuring QoS Profiles with XML (p. 971)

6.283.2.7 datawriter_qos

```
const struct DDS_DataWriterQos* RTI_Connext_RequesterParams::datawriter_qos
```

Sets the quality of service of the request DataWriter.

See also

RTI_Connext_RequesterParams::qos_profile_name (p. 2175)

6.283.2.8 datareader_qos

```
const struct DDS_DataReaderQos* RTI_Connext_RequesterParams::datareader_qos
```

Sets the quality of service of the request DataReader.

See also

RTI_Connext_RequesterParams::qos_profile_name (p. 2175)

6.283.2.9 publisher

```
DDS_Publisher* RTI_Connext_RequesterParams::publisher
```

Sets a specific Publisher.

By default, a Requester uses the DomainParticipant's implicit Publisher. Sometimes a different Publisher may be needed, for example, to use non-default PublisherQos.

6.283.2.10 subscriber

```
DDS_Subscriber* RTI_Connext_RequesterParams::subscriber
```

Sets a specific Subscriber.

By default, a Requester uses the DomainParticipant's implicit Subscriber. Sometimes a different Subscriber may be needed, for example, to use non-default SubscriberQos.

6.284 RTI_Connext_SimpleReplierListener Struct Reference

The listener called by a SimpleReplier.

Data Fields

- **RTI_Connext_SimpleReplierListener_OnRequestAvailableCallback** `on_request_available`
User callback that receives a request and provides a reply.
- **RTI_Connext_SimpleReplierListener_OnReturnLoanCallback** `return_loan`
Returns a previously generated reply to the user.
- `void *` **user_data**
User data attached to a listener.

6.284.1 Detailed Description

The listener called by a SimpleReplier.

See also

FooBarSimpleReplier (p. 2112)

SimpleReplier example (p. 1009)

6.284.2 Field Documentation

6.284.2.1 user_data

```
void* RTI_Connext_SimpleReplierListener::user_data
```

User data attached to a listener.

6.285 RTI_Connext_SimpleReplierParams Struct Reference

Contains the parameters for creating a **FooBarSimpleReplier** (p. 2112).

Data Fields

- **DDS_DomainParticipant * participant**
The participant this replier uses to join a DDS domain.
- **char * service_name**
The service name the Replier offers and Requesters use to match.
- **char * request_topic_name**
Sets a specific request topic name.
- **char * reply_topic_name**
Sets a specific reply topic name.
- **char * qos_library_name**
Sets the library that contains the `RTI_Connext_SimpleReplierParams::qos_profile_name` (p. 2179).
- **char * qos_profile_name**
Sets a QoS profile for the entities in this replier.
- **const struct DDS_DataWriterQos * datawriter_qos**
Sets the quality of service of the reply DataWriter.
- **const struct DDS_DataReaderQos * datareader_qos**
Sets the quality of service of the request DataReader.
- **DDS_Publisher * publisher**
Sets a specific Publisher.
- **DDS_Subscriber * subscriber**
Sets a specific Subscriber.
- **struct RTI_Connext_SimpleReplierListener * simple_listener**
Sets a listener that is called when requests are available.

6.285.1 Detailed Description

Contains the parameters for creating a **FooBarSimpleReplier** (p. 2112).

The parameters for a SimpleReplier are identical to those of the Replier, except for the SimpleReplierListener, which is required and has a different user callback.

See also

RTI_Connext_ReplierParams (p. 2169)

6.285.2 Field Documentation

6.285.2.1 participant

```
DDS_DomainParticipant* RTI_Connext_SimpleReplierParams::participant
```

The participant this replier uses to join a DDS domain.

6.285.2.2 service_name

```
char* RTI_Connext_SimpleReplierParams::service_name
```

The service name the Replier offers and Requesters use to match.

See also

RTI_Connext_RequesterParams::service_name (p. 2174)

6.285.2.3 request_topic_name

```
char* RTI_Connext_SimpleReplierParams::request_topic_name
```

Sets a specific request topic name.

This is an alternative to **RTI_Connext_SimpleReplierParams::service_name** (p. 2178)

6.285.2.4 reply_topic_name

```
char* RTI_Connext_SimpleReplierParams::reply_topic_name
```

Sets a specific reply topic name.

This is an alternative to **RTI_Connext_SimpleReplierParams::service_name** (p. 2178)

6.285.2.5 qos_library_name

```
char* RTI_Connext_SimpleReplierParams::qos_library_name
```

Sets the library that contains the **RTI_Connext_SimpleReplierParams::qos_profile_name** (p. 2179).

See also

RTI_Connext_ReplierParams::qos_profile_name (p. 2171)

6.285.2.6 qos_profile_name

```
char* RTI_Connext_SimpleReplierParams::qos_profile_name
```

Sets a QoS profile for the entities in this replier.

See also

RTI_Connext_ReplierParams::qos_profile_name (p. 2171)

6.285.2.7 datawriter_qos

```
const struct DDS_DataWriterQos* RTI_Connext_SimpleReplierParams::datawriter_qos
```

Sets the quality of service of the reply DataWriter.

See also

RTI_Connext_ReplierParams::qos_profile_name (p. 2171)

6.285.2.8 datareader_qos

```
const struct DDS_DataReaderQos* RTI_Connext_SimpleReplierParams::datareader_qos
```

Sets the quality of service of the request DataReader.

See also

RTI_Connext_ReplierParams::qos_profile_name (p. 2171)

6.285.2.9 publisher

```
DDS_Publisher* RTI_Connext_SimpleReplierParams::publisher
```

Sets a specific Publisher.

See also

RTI_Connext_RequesterParams::publisher (p. 2176)

6.285.2.10 subscriber

DDS_Subscriber* RTI_Connext_SimpleReplierParams::subscriber

Sets a specific Subscriber.

See also

RTI_Connext_RequesterParams::subscriber (p. 2176)

6.285.2.11 simple_listener

struct RTI_Connext_SimpleReplierListener* RTI_Connext_SimpleReplierParams::simple_listener

Sets a listener that is called when requests are available.

See also

RTI_Connext_ReplierListener (p. 2168)

6.286 TransportAllocationSettings_t Struct Reference

Allocation settings used by various internal buffers.

```
#include <transport_common_user.h>
```

6.286.1 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.

Chapter 7

File Documentation

7.1 howto_prototypes.h

```
1  /* $Id$
2
3  (c) Copyright, Real-Time Innovations, 2004-2016.
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 1.0d,19apr04,eyr Read/take parameters changed
16 1.0a,27oct03,gpc Written by combining pre-existing headers
17 ===== */
18
19 #include "Foo.h"
20 #include "FooSupport.h"
21
22
23 extern DDS_DomainParticipantFactory* howto_get_factory(void);
24
25 extern DDS_DomainParticipant*
26     howto_setup_participant(DDS_DomainParticipantFactory* factory,
27                             DDS_DomainId_t domain_id);
28
29 extern void howto_teardown_participant(DDS_DomainParticipantFactory* factory,
30                                         DDS_DomainParticipant* participant);
31
32 extern void howto_register_type(DDS_DomainParticipant* participant,
33                                 const char* type_name);
34
35 extern DDS_Topic* howto_setup_topic(DDS_DomainParticipant* participant,
36                                     const char* topic_name, const char* type_name);
37
38 extern void howto_teardown_topic(DDS_DomainParticipant* participant,
39                                  DDS_Topic* topic);
40
41 extern DDS_DataWriter* howto_setup_writer(DDS_Publisher* publisher,
42                                             DDS_Topic* topic);
43
44
45 extern void howto_send_data(FooDataWriter* writer, Foo* data,
46                             DDS_InstanceHandle_t instance_handle);
47
48 extern void howto_teardown_writer(DDS_Publisher* publisher,
49                                   DDS_DataWriter* writer);
50
51 extern DDS_DataReader* howto_setup_reader(DDS_Subscriber* subscriber,
```

```

52             DDS_Topic* topic);
53
54 extern void howto_teardown_reader(DDS_Subscriber* subscriber,
55                                   DDS_DataReader* reader);
56
57 extern void howto_setup_reader_to_access_received_data_via_listener(
58     DDS_DataReader* entity, struct DDS_DataReaderListener* entity_listener);
59
60 extern void howto_access_received_data_via_reader(
61     DDS_DataReader* 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 DDS_Subscriber* howto_setup_subscriber(DDS_DomainParticipant* participant);
68
69 extern void howto_teardown_subscriber(DDS_DomainParticipant* participant,
70                                       DDS_Subscriber* subscriber);
71
72 extern void howto_setup_subscriber_to_access_received_data_via_listener(
73     DDS_Subscriber* entity, struct DDS_SubscriberListener* entity_listener);
74
75 extern void howto_access_received_data_via_subscriber(DDS_Subscriber* subscriber);
76
77 extern DDS_Publisher* howto_setup_publisher(DDS_DomainParticipant* participant);
78
79 extern void howto_teardown_publisher(DDS_DomainParticipant* participant,
80                                       DDS_Publisher* publisher);

```

7.2 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 /*
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
387 extern NDDS_Transport_DllExport RTIBool NDDS_Transport_get_address(
388     char *addresses_str_out,
389     size_t addresses_str_buffer_size_in,
390     const char *list,
391     RTI_UINT32 index);
392
393 extern NDDS_Transport_DllExport RTIBool
394 NDDS_Transport_get_number_of_addresses_in_string(
395     RTI_UINT32 *number_out,
396     const char *list);
397
398 #ifdef __cplusplus
399 } /* extern "C" */
400 #endif
401
402 #include "transport/transport_common_impl.h"
403
404 #endif /* transport_common_h */

```

7.3 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 #include
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 Connex 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 Connex 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. leftmost) \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_UINT32 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},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 */

```

7.4 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 RTI_Boolean 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 structure 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_FALSE 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 */

```


7.5 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 shmемUuid will be the same as the transport_uuid.
100     */
101     RTI_Bool reserved;
102
103     /*i
104      * UUID used to check the reachability of a shmем locator received through
105      * discovery. After attaching to a shmем 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 shmем.
109      */
110     NDDS_Transport_Address_t shmемUUID;
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 shmемUuid 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_LOADED)
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, 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 RTI_Bool
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     RTI_Bool *shmem_segment_exists,
381     RTI_Bool *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 */

```

7.6 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 /* ===== */
46
47 typedef void (
48     *NDDS_Transport_UDP_WAN_Listener_Public_Mapping_Resolved_Callback) (
49     void *listener_data,
50     const NDDS_Transport_Address_t *public_uuid_address,
51     NDDS_Transport_Port_t rtps_port);
52
53 struct NDDS_Transport_UDP_WAN_Listener {
54     void *listener_data;
55     NDDS_Transport_UDP_WAN_Listener_Public_Mapping_Resolved_Callback
56         on_public_mapping_resolved;
57 };
58
59 /* ===== */
60 /* UDP Port */
61 /* ===== */
62
63 /*e \dref_UDPTransport_Port
64 */
65
66 typedef RTI_UINT16 NDDS_Transport_UDP_Port;
67
68 #define NDDS_TRANSPORT_UDP_PORT_INVALID (0)
69 #define NDDS_TRANSPORT_UDP_PORT_MAX (RTI_UINT16_MAX)
70
71 /* ===== */
72 /* UDP WAN CommPorts */
73 /* ===== */
74
75 /*e \dref_UDPTransport_WAN_CommPortsMappingInfo
76 */
77
78 struct NDDS_Transport_UDP_WAN_CommPortsMappingInfo {
79     /*e \dref_UDPTransport_WAN_CommPortsMappingInfo_rtps_port
80     */
81     NDDS_Transport_Port_t rtps_port;
82     /*e \dref_UDPTransport_WAN_CommPortsMappingInfo_host_port
83     */
84     NDDS_Transport_UDP_Port host_port;
85     /*e \dref_UDPTransport_WAN_CommPortsMappingInfo_public_port
86     */
87     NDDS_Transport_UDP_Port public_port;
88 };
89
90 /* ===== */
91 /* Unicast UDP Properties */
92 /* ===== */

```

```

98 /* ===== */
99
100 struct NDDS_Transport_UDP_Property_t {
101     struct NDDS_Transport_Property_t parent;
102     RTI_INT32 send_socket_buffer_size;
103     RTI_INT32 recv_socket_buffer_size;
104     RTI_INT32 unicast_enabled;
105     RTI_INT32 multicast_enabled;
106     RTI_INT32 multicast_ttl;
107     RTI_INT32 multicast_loopback_disabled;
108     RTI_INT32 ignore_loopback_interface;
109     RTI_INT32 ignore_nonup_interfaces;
110     RTI_INT32 ignore_nonrunning_interfaces;
111     RTI_INT32 no_zero_copy;
112     RTI_INT32 send_blocking;
113     RTI_INT32 enable_v4mapped;
114     RTI_INT32 use_checksum;
115     RTI_UINT32 transport_priority_mask;
116     RTI_INT32 transport_priority_mapping_low;
117     RTI_INT32 transport_priority_mapping_high;
118     RTI_INT32 send_ping;
119     RTI_INT32 force_interface_poll_detection;
120     RTI_UINT32 interface_poll_period;
121     RTI_INT32 reuse_multicast_receive_resource;
122     RTI_INT32 protocol_overhead_max;
123     RTI_INT32 disable_interface_tracking;
124     RTI_UINT32 join_multicast_group_timeout;
125     char *public_address;
126     struct NDDS_Transport_UDP_WAN_CommPortsMappingInfo *comm_ports_list;
127     RTI_INT32 comm_ports_list_length;
128     struct NDDS_Transport_UDP_WAN_Listener *wan_listener;
129     RTI_INT32 port_offset;
130     RTI_UINT32 binding_ping_period;
131     /*i
132      * \brief If this property is set, we use this function to check for an
133      * enabled WAN transport. If the function is not set, we would try to load a
134      * default shared library with the correct function to enable WAN transport.
135      */
136     NDDS_Transport_UDP_WAN_Library_Is_Plugin_Enabled_Function
137         plugin_enabled_function_ptr;
138     /****** START TESTING ONLY FIELDS *****/
139     /*i
140      * \brief If this property is set we will update the send
141      * and receive statistics split into participant and endpoint discovery
142      * by parsing the packets. We will update the discovery/user packets
143      * based on the port number regardless of if this property is set.
144      * The default is false for performance reasons and is set to true only
145      * for testing purposes.
146      */
147     RTIBool gather_detailed_statistics;
148     /*i
149      * \brief If this property is set to TRUE we will check if the guid prefix
150      * in the RTPS header matches guid_prefix_to_drop. If it does we will drop
151      * the sample.
152      */
153     RTIBool drop_samples_by_guid_prefix;
154     /*i
155      * \brief GUID prefix to drop when drop_samples_by_guid_prefix is set to
156      * TRUE.
157      */
158     struct RTIOsapiRtpsGuidPrefix guid_prefix_to_drop;
159     /****** END TESTING ONLY FIELDS *****/
160 };
161
162 /*e \ingroup NDDS_Transport_UDP_Plugin
163  * \brief Default value of NDDS_Transport_Property_t#properties_bitmap.
164  */
165 #define NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT (0)
166
167 /*e \ingroup NDDS_Transport_UDP_Plugin
168  * \brief Default value of
169  * NDDS_Transport_Property_t#gather_send_buffer_count_max.
170  *
171  * This is also the maximum value that can be used when instantiating the udp
172  * transport.
173  *
174  * 16 is sufficient for \ndds, but more may improve discovery and reliable
175  * performance.
176  * Porting note: find out what the maximum gather buffer count is on your OS!
177  */
178 #define NDDS_TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT (16)

```

```

179
180 /*e \ingroup NDDS_Transport_UDP_Plugin
181 * \brief Used to specify that os maximum be used to specify socket buffer size.
182 *
183 * This property, if set, uses the maximum allowable size set in your OS for
184 * send and receive socket buffer sizes. If you update the maximum allowable
185 * size in your OS setting, then \ndds will automatically pick up that change.
186 * This property is not supported on all platforms because not all platforms
187 * have maximum values for the size of the send and receive buffers for sockets.
188 * If the platform does not support this property, then the property will be
189 * set to \ref
190 * NDDS_Transport_UDP_Plugin#NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT
191 * or \ref
192 * NDDS_Transport_UDP_Plugin#NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT,
193 * depending on the property being set.
194 * See the Platform Notes to determine if this property is supported on your
195 * platform.
196 *
197 * This value is -2.
198 */
199 #define NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_MAX \
200     RTI_OSAPI_UDP_SOCKET_BUFFER_SIZE_OS_MAX
201
202 /*e \ingroup NDDS_Transport_UDP_Plugin
203 * \brief Used to specify that os default be used to specify socket buffer size.
204 *
205 * This property, if set, uses the default size set in your OS for send and
206 * receive socket buffer sizes. If you update the default size in your OS
207 * settings, then \ndds will automatically pick up that change.
208 *
209 * This value is -1.
210 */
211 #define NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT \
212     RTI_OSAPI_UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT
213
214 /*e \ingroup NDDS_Transport_UDP_Plugin
215 * \brief Default value of send_socket_buffer_size.
216 *
217 * The default value is 2097152 bytes (1048576 bytes on AIX platforms).
218 */
219 #define NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT \
220     RTI_OSAPI_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT
221
222 /*e \ingroup NDDS_Transport_UDP_Plugin
223 * \brief Default value of recv_socket_buffer_size.
224 *
225 * The default value is 2097152 bytes (1048576 bytes on AIX platforms).
226 */
227 #define NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT \
228     RTI_OSAPI_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT
229
230 #define NDDS_TRANSPORT_UDP_DATAGRAM_LENGTH_MAX (65535)
231
232 /*e \ingroup NDDS_Transport_UDP_Plugin
233 * \brief Default value of multicast_ttl.
234 */
235 #define NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT (1)
236
237
238 #ifdef RTI_MULTICAST
239     #define NDDS_TRANSPORT_UDP_USE_MULTICAST_DEFAULT 1
240 #else
241     #define NDDS_TRANSPORT_UDP_USE_MULTICAST_DEFAULT 0
242 #endif
243
244
245 /*e \dref_UDPTransport_IGNORE_LOOPBACK_DISABLED
246 */
247 #define NDDS_TRANSPORT_UDP_IGNORE_LOOPBACK_DISABLED (0)
248
249 /*e \dref_UDPTransport_IGNORE_LOOPBACK_ENABLED
250 */
251 #define NDDS_TRANSPORT_UDP_IGNORE_LOOPBACK_ENABLED (1)
252
253 /*e \dref_UDPTransport_IGNORE_LOOPBACK_AUTO
254 */
255 #define NDDS_TRANSPORT_UDP_IGNORE_LOOPBACK_AUTO (-1)
256
257
258 /*e \dref_UDPTransport_BLOCKING_NEVER
259 */

```



```

260 #define NDDS_TRANSPORT_UDP_BLOCKING_NEVER (0)
261
262 /*e \dref_UDPTransport_BLOCKING_ALWAYS
263 */
264 #define NDDS_TRANSPORT_UDP_BLOCKING_ALWAYS (1)
265 #define NDDS_TRANSPORT_UDP_BLOCKING_UNICAST_ONLY (2)
266
267 /*e \dref_UDPTransport_BLOCKING_DEFAULT
268 */
269 #define NDDS_TRANSPORT_UDP_BLOCKING_DEFAULT NDDS_TRANSPORT_UDP_BLOCKING_ALWAYS
270
271 /* ===== */
272 /*           InterfaceListener Class           */
273 /* ===== */
274
275 /*i
276 * \defgroup NDDS_Transport_UDP_InterfaceListenerClass
277 * NDDS_Transport_UDP_InterfaceListener \ingroup NDDS_Transport_UDP_Plugin
278 * \brief Interface for IPv4 or v6 interface acceptance/rejection
279 *
280 * @mtsafety See NDDS_Transport_UDP_public UML diagram
281 *
282 * @see NDDS_Transport_UDP_newI
283 */
284 struct NDDS_Transport_UDP_InterfaceListener; /*forward declare */
285
286 /*i \ingroup NDDS_Transport_UDP_InterfaceListenerClass
287 * \brief A user-provided function to be used to filter out particular
288 * network interfaces to be used by the Transport Plugin.
289 *
290 * The user may implement this function and pass it to an
291 * instance of a Transport Plugin to reject or accept network
292 * interfaces that the plugin offers as ones to use to receive
293 * messages.
294 *
295 * When \ndds contacts other applications, it will provide a list of
296 * interfaces that other applications can use to send messages (on a
297 * plugin by plugin basis). The addresses of the interfaces are used
298 * as the destination addresses of messages that are sent. If a plugin
299 * rejects one of the available network interfaces, then that interface
300 * will not be offered as one to which other plugins should send
301 * messages.
302 *
303 * For interfaces that can be used by the plugin, this function also
304 * enables the user to select which interfaces \ndds is allowed to use
305 * in sending multicast messages. The set of interfaces enabled for
306 * multicast sending is a subset of the interfaces that can be used for
307 * receiving. That is, an interface cannot be enabled to send
308 * multicast messages but disabled for receiving unicast messages.
309 *
310 * Only IP network interfaces that are "UP" (enabled by the OS and
311 * ready to use) will be passed to this filter function. The ones that
312 * are "DOWN" will not be used by definition.
313 *
314 * @param plugin \st_in The instance of the plugin for which the filter
315 * is being called.
316 *
317 * @param multicast_enabled_out \st_out The filter should set whether
318 * or not the plugin is allowed to use this interface for
319 * sending/receiving multicast messages.
320 *
321 * @param interface_in \st_in An interface offered by the plugin as a
322 * candidate for receive messages.
323 *
324 * @return 1 to use the interface.
325 *         \c multicast_enabled_out will indicate whether it is also OK
326 *         to use the interface to send multicast messages.
327 *
328 * @return 0 to reject using the interface.
329 */
330 typedef RTI_INT32 (*NDDS_Transport_UDP_Interface_Filter_Fcn)(
331     struct NDDS_Transport_UDP_InterfaceListener *self,
332     struct NDDS_Transport_UDP *plugin,
333     RTI_INT32 *multicast_enabled_out,
334     NDDS_Transport_Interface_t *interface_in,
335     const char *interface_name_in);
336
337 /*i \ingroup NDDS_Transport_UDP_InterfaceListenerClass
338 * \brief Pure virtual class for the interface listener.
339 * RTI does NOT supply a default interface listener
340 */

```

```

341 struct NDDS_Transport_UDP_InterfaceListener {
342     /* \copydoc NDDS_Transport_UDP_Interface_Filter_Fcn */
343     NDDS_Transport_UDP_Interface_Filter_Fcn onInterface;
344 };
345
346 /* ===== */
347 /*           Socket Factory Class           */
348 /* ===== */
349 /*i
350 * \defgroup NDDS_Transport_UDP_SocketFactoryClass
351 * NDDS_Transport_UDP_SocketFactory \ingroup NDDS_Transport_UDP_Plugin \brief
352 * Interface for socket creation and destruction.
353 *
354 * @mtsafety See NDDS_Transport_UDP_public UML diagram
355 *
356 * @see NDDS_Transport_UDP_new
357 */
358
359 /*i
360 * \ingroup NDDS_Transport_UDP_SocketFactoryClass
361 * \brief Interface representation for socket creation and destruction.
362 */
363 struct NDDS_Transport_UDP_SocketFactory;
364
365 /*i
366 * \ingroup NDDS_Transport_UDP_SocketFactoryClass
367 * \brief Create a new sending socket for the Itransport.
368 *
369 * @pre a valid UDP transport.
370 *
371 * @post A valid UDP send socket, that may not necessarily be bound.
372 *
373 * @param transport_in \b In. The NDDS_Transport_UDP transport that will use
374 * the new socket. Specific property of the transport, such as the maximum
375 * message size allowed, can be queried from the
376 * NDDS_Transport_Plugin#property
377 *
378 * @param interface_address_in \b In. If set, must be the address of
379 * a valid interface. In addition, the socket created should be enabled
380 * to send via multicast. If is NULL, then the socket created will not
381 * be enabled to send multicast.
382 *
383 * @param interface_index_in \b In. If set, must be the index of
384 * a valid interface. In addition, the socket created should be enabled
385 * to send via multicast. If is NULL, then the socket created will not
386 * be enabled to send multicast. Interface index is used instead of an
387 * interface address in UDPv6.
388 *
389 * @return Newly created socket descriptor.
390 *
391 * @mtsafety «oneThreadAtATime»
392 *
393 * @see NDDS_Transport_UDP_SocketFactory_create_send_socket
394 * NDDS_Transport_UDP_SocketFactory_Socket_Destroy_Fcn
395 */
396 typedef RTI_INT32 (*NDDS_Transport_UDP_SocketFactory_Send_Socket_Create_Fcn) (
397     struct NDDS_Transport_UDP_SocketFactory *me,
398     NDDS_Transport_Port_t *port_out,
399     const struct NDDS_Transport_UDP *transport_in,
400     const NDDS_Transport_Address_t *interface_address_in,
401     const int *interface_index_in);
402
403 /*i
404 * \ingroup NDDS_Transport_UDP_SocketFactoryClass
405 * \brief Create a new receiving socket for the transport, which
406 * may be a multicast transport.
407 *
408 * @pre valid IP unicast transport.
409 *
410 * @post A valid, bound unicast receive socket.
411 *
412 * @param port_inout. \b InOut. NOT optional. If
413 * NDDS_TRANSPORT_PORT_INVALID, a new, previously unassigned port must
414 * be chosen, and the port must be assigned a new value so the caller
415 * knows which port it is listening on. If the port is a valid value,
416 * you must attempt to bind to that port, and not proceed in case of
417 * failure (such as if the port is already bound).
418 *
419 * @param transport_in \b In. The transport that wishes to use the
420 * socket. Properties of the IP unicast transport described in
421 * NDDS_Transport_UDP_Property_t must be met in "at least" manner. That

```

```

422 * is, doing more than requested is allowed (but will lead to waste
423 * unless the implementer has other insights which suggests more
424 * optimal use of resources), but creation of a socket that cannot meet
425 * the minimum requirement will put the system into an inconsistent
426 * state. For example, the receiving socket will be either polled or
427 * blocking; if the transport is polled
428 * (NDDS_Transport_Plugin_isPolled) it is an error to create a blocking
429 * socket. On the other hand, creating a socket that can buffer more
430 * data than requires is allowed. Specific property of the transport,
431 * such as the maximum message size to be buffered, can be queried from
432 * the NDDS_Transport_Plugin#property.
433 *
434 * @param transport_priority_in \b In. transport_priority hint. The
435 * RTI supplied socket factory NDDS_Transport_UDP_SocketFactory does not
436 * act on transport_priority. If you want to implement a
437 * transport_priority aware socket factory, either implement your own
438 * from scratch or use
439 * NDDS_Transport_UDP_SocketFactory_createReceiveSocket and specialize.
440 *
441 * @param multicast_address_in \b In. Valid if joining a multicast group.
442 *
443 * @see NDDS_Transport_UDP_SocketFactory_Send_Socket_Create_Fcn
444 * NDDS_Transport_UDP_SocketFactory_create_receive_socket
445 * NDDS_Transport_UDP_SocketFactory_Socket_Destroy_Fcn
446 */
447 typedef RTI_INT32 (*NDDS_Transport_UDP_SocketFactory_Receive_Socket_Create_Fcn) (
448     struct NDDS_Transport_UDP_SocketFactory *me,
449     NDDS_Transport_Port_t *port_inout,
450     const struct NDDS_Transport_UDP *transport_in,
451     const NDDS_Transport_Address_t *multicast_address_in);
452
453 /*i \ingroup NDDS_Transport_UDP_SocketFactoryClass
454 * \brief Destroy a previously created socket.
455 *
456 * @pre Socket no longer used, and no thread blocked on the socket.
457 * @post All resources associate with the socket released.
458 *
459 * @param s_in \b In. socket descriptor.
460 *
461 * @mtsafety <oneThreadAtATime>
462 *
463 * @see NDDS_Transport_UDP_SocketFactory_destroy_socket
464 */
465 typedef RTI_INT32 (*NDDS_Transport_UDP_SocketFactory_Socket_Destroy_Fcn) (
466     struct NDDS_Transport_UDP_SocketFactory *me,
467     RTI_INT32 s_in);
468
469 /*i \ingroup NDDS_Transport_UDP_SocketFactoryClass
470 * \brief Pure virtual class for the socket factory functionality.
471 * RTI supplies the UDP socket factory.
472 */
473 struct NDDS_Transport_UDP_SocketFactory {
474     /*e Children: implement this method. */
475     NDDS_Transport_UDP_SocketFactory_Send_Socket_Create_Fcn create_send_socket;
476
477     /*e Children: implement this method. */
478     NDDS_Transport_UDP_SocketFactory_Receive_Socket_Create_Fcn
479         create_receive_socket;
480
481     /*e Children: implement this method to destroy both the send and
482     receive sockets. */
483     NDDS_Transport_UDP_SocketFactory_Socket_Destroy_Fcn destroy_socket;
484 };
485
486 /*i \ingroup NDDS_Transport_UDP_SocketFactoryClass
487 * \brief Convenience to check if the pure virtual methods have all been
488 * implemented
489 */
490 #define NDDS_Transport_UDP_SocketFactory_is_valid(m) \
491     ((m)->create_send_socket && (m)->create_receive_socket && (m)->destroy_socket)
492
493 /* ===== */
494 /* IP Socket Factory Class */
495 /* ===== */
496
497 /*i \ingroup NDDS_Transport_UDP_Plugin
498 * \brief Realization of NDDS_Transport_Get_Class_Name_Fcn_cEA for IPv6
499 * transports.
500 *
501 * Returns the string "udp4" or "udp6".
502 */

```

```

503 * @see NDDS_Transport_Get_Class_Name_Fcn_cEA for complete
504 * documentation.
505 */
506 extern NDDS_Transport_DllExport const char *
507 NDDS_Transport_UDP_get_class_name_cEA(NDDS_Transport_Plugin *self);
508
509 /*i \ingroup NDDS_Transport_UDP_Plugin
510 * \brief Realization of NDDS_Transport_String_To_Address_Fcn_cEA for IP
511 * transports.
512 *
513 * Converts a host name string to an address.
514 *
515 * @param self NOT USED. May be NULL.
516 *
517 * @param address_out \st_out The corresponding numerical value in IPv6
518 * format.
519 *
520 * @param address_in \st_in The name of the address. It can be a
521 * dot notation name or a host name. If NULL, then the IP address of
522 * the localhost will be returned.
523 *
524 * @see NDDS_Transport_String_To_Address_Fcn_cEA for complete
525 * documentation.
526 */
527 extern NDDS_Transport_DllExport RTI_INT32
528 NDDS_Transport_UDP_string_to_address_cEA(
529     NDDS_Transport_Plugin *self,
530     NDDS_Transport_Address_t *address_out,
531     const char *address_in);
532
533 /* ===== */
534 /* UDP/IP Transport Plugin */
535 /* ===== */
536
537 /*i \ingroup NDDS_Transport_UDP_Plugin
538 * \brief Create an instance of a UDP Transport Plugin.
539 *
540 * An application may create and register multiple instances of this
541 * Transport Plugin with \ndds. Multiple instances of the plugin may be useful
542 * for partitioning the network interfaces across multiple \ndds domains.
543 * However, note that the underlying transport, the operating system's IP layer,
544 * is still a "singleton." For example, if a unicast transport has already
545 * bound to a port, and another unicast transport tries to bind to the same
546 * port, the second attempt will fail.
547 *
548 * The transport plugin honors the interface/multicast "white" and "black"
549 * lists specified in the NDDS_Transport_UDP_Property_t#parent:
550 * - NDDS_Transport_Property_t#allow_interfaces_list,
551 * - NDDS_Transport_Property_t#deny_interfaces_list,
552 * - NDDS_Transport_Property_t#allow_multicast_interfaces_list,
553 * - NDDS_Transport_Property_t#deny_multicast_interfaces_list
554 *
555 * The format of a string in these lists is assumed to be in standard IPv4
556 * dot notation, possibly containing wildcards. For example:
557 * - 10.10.30.232
558 * - 10.10.*.*
559 * - 192.168.1.*
560 * - etc.
561 * Strings not in the correct format will be ignored.
562 *
563 * @param property_in \st_in Desired behavior of this transport. May be
564 * NULL for default property.
565 *
566 * @return A UDP Transport Plugin instance on success; or NULL on failure.
567 */
568 extern NDDS_Transport_DllExport NDDS_Transport_Plugin *NDDS_Transport_UDP_new(
569     const struct NDDS_Transport_UDP_Property_t *property_in);
570
571 /*i \ingroup NDDS_Transport_UDP_Plugin
572 * \brief Create an instance of a UDP Transport Plugin.
573 *
574 * \copydoc NDDS_Transport_UDP_new
575 *
576 * @param socket_factory_in \st_in The factory that will create all the
577 * sockets to be used in the transport. May be NULL if default factory
578 * is to be used. Only used by RTI internal testing.
579 * End users should set this parameter to NULL.
580 *
581 * @param interface_filter_in \st_in A function that will be used to
582 * filter out network interfaces that this plugin can use. May be NULL if
583 * default interface listener is to be used. The default listener uses

```

```

584 * The NDDS_Transport_UDP_Property_t#parent's lists to select the interfaces.
585 * Only used by RTI internal testing. End users should set this
586 * parameter to NULL.
587 *
588 * @exception RTI_LOG_MALLOC_FAILURE_d RTI_LOG_CREATION_FAILURE_s
589 * RTI_LOG_PRECONDITION_FAILURE RTI_LOG_INIT_FAILURE_s
590 */
591 extern NDDS_Transport_DllExport NDDS_Transport_Plugin *NDDS_Transport_UDP_newI(
592     const struct NDDS_Transport_UDP_Property_t *property_in,
593     struct NDDS_Transport_UDP_SocketFactory *socket_factory_in,
594     struct NDDS_Transport_UDP_InterfaceListener *interface_listener_in);
595
596 /*i \ingroup NDDS_Transport_UDP_Plugin
597 * \brief Realization of NDDS_Transport_Get_Receive_Interfaces_Fcn_cEA
598 * for IP transports.
599 *
600 * Retrieves a list of available network interfaces.
601 *
602 * @see NDDS_Transport_Get_Receive_Interfaces_Fcn_cEA for complete
603 * documentation.
604 */
605 extern NDDS_Transport_DllExport RTI_INT32
606 NDDS_Transport_UDP_get_receive_interfaces_cEA(
607     NDDS_Transport_Plugin *self,
608     RTI_INT32 *found_more_than_provided_for_out,
609     RTI_INT32 *interface_reported_count_out,
610     NDDS_Transport_Interface_t interface_array_inout[],
611     RTI_INT32 interface_array_size_in);
612
613 /*i \ingroup NDDS_Transport_UDP_Plugin
614 * \brief Retrieve ignore_loopback property
615 */
616 extern NDDS_Transport_DllExport RTI_INT32
617 NDDS_Transport_UDP_get_ignore_loopback_interface_status(
618     const NDDS_Transport_Plugin *self);
619
620 /*i \ingroup NDDS_Transport_UDP_Plugin
621 * \brief Checks if the provided address is a loopback address or a known
622 * interface address.
623 *
624 * The provided address is an IPv6 address which is internally mapped into a
625 * IPv4 socket address (network byte order).
626 *
627 * Unions are used to force 4-byte alignment of the IPV4 portion of
628 * the addresses. Otherwise we risk bus faults if we are ever passed
629 * unaligned address arrays.
630 */
631 extern NDDS_Transport_DllExport RTI_Bool NDDS_Transport_UDP_is_local_address(
632     const NDDS_Transport_Plugin *self,
633     const NDDS_Transport_Address_t *address_in);
634
635 /*i \ingroup NDDS_Transport_UDP_Plugin
636 * Map transport_priority to an IP_TOS value based on the configured
637 * transport properties.
638 *
639 * @brief transport_priority to IP_TOS mapping
640 */
641 extern NDDS_Transport_DllExport RTI_INT32
642 NDDS_Transport_UDP_map_transport_priority(
643     struct NDDS_Transport_UDP_Property_t *property,
644     RTI_INT32 transport_priority_in);
645
646 /*i \ingroup NDDS_Transport_UDP_Plugin
647 * \brief Size of special "unblock" message
648 *
649 * When an ReceiveResource is unblocked with
650 * NDDS_Transport_unblock_recv_fnc_rrEA, a special unblock
651 * message may be sent. This message is guaranteed to be smaller than the
652 * minimum RTPS message size. Higher layer that processes messages
653 * returned by netio should throw away messages of this size, as it was
654 * intended to unblock a blocked thread only.
655 */
656 #define NDDS_TRANSPORT_UDP_UNBLOCK_MESSAGE_SIZE \
657     (4 + 4 + sizeof(NDDS_Transport_Plugin *))
658
659 /*i \ingroup NDDS_Transport_UDP_Plugin
660 * \brief DiffServ max value
661 *
662 * Maximum value of the DiffServ field in an IP packet
663 */
664 #define NDDS_TRANSPORT_UDP_DIFF_SERV_MAX_VALUE RTI_OSAPI_SOCKET_MAX_DIFF_SERV

```

```

665
666 struct NDDS_Transport_UDP_Statistics {
667     /*****
668      * PUBLIC FIELDS
669      *****/
670     RTI_ATOMIC(RTI_UINT64) user_bytes_sent;
671     RTI_ATOMIC(RTI_UINT64) user_bytes_received;
672     RTI_ATOMIC(RTI_UINT64) user_packets_sent;
673     RTI_ATOMIC(RTI_UINT64) user_packets_received;
674     /*
675      * discovery_bytes and discovery_packets are calculated using the
676      * port number that is sent to/received from. This assumes the port
677      * is an even number. When using an odd port number (ie, using an odd
678      * domain ID), these statistics will not be calculated correctly.
679      * You should not use these members in tests where the domain ID
680      * and port numbers may change.
681      */
682     RTI_ATOMIC(RTI_UINT64) discovery_bytes_sent;
683     RTI_ATOMIC(RTI_UINT64) discovery_bytes_received;
684     RTI_ATOMIC(RTI_UINT64) discovery_packets_sent;
685     RTI_ATOMIC(RTI_UINT64) discovery_packets_received;
686     /*
687      * participant_discovery_bytes and participant_discovery_packets refer to
688      * DATA(P)s or participant bootstrap messages (i.e. messages sent/received
689      * by the anonymous participant writer/reader).
690      */
691     RTI_ATOMIC(RTI_UINT64) participant_discovery_bytes_sent;
692     RTI_ATOMIC(RTI_UINT64) participant_discovery_bytes_received;
693     RTI_ATOMIC(RTI_UINT64) participant_discovery_packets_sent;
694     RTI_ATOMIC(RTI_UINT64) participant_discovery_packets_received;
695     /*
696      * participant_config_bytes and participant_config_packets refer to
697      * participant configuration announcements sent reliably between
698      * participants.
699      */
700     RTI_ATOMIC(RTI_UINT64) participant_config_bytes_sent;
701     RTI_ATOMIC(RTI_UINT64) participant_config_bytes_received;
702     RTI_ATOMIC(RTI_UINT64) participant_config_packets_sent;
703     RTI_ATOMIC(RTI_UINT64) participant_config_packets_received;
704     RTI_ATOMIC(RTI_UINT64) automatic_liveliness_bytes_sent;
705     RTI_ATOMIC(RTI_UINT64) automatic_liveliness_bytes_received;
706     RTI_ATOMIC(RTI_UINT64) automatic_liveliness_packets_sent;
707     RTI_ATOMIC(RTI_UINT64) automatic_liveliness_packets_received;
708     RTI_ATOMIC(RTI_UINT64) manual_liveliness_bytes_sent;
709     RTI_ATOMIC(RTI_UINT64) manual_liveliness_bytes_received;
710     RTI_ATOMIC(RTI_UINT64) manual_liveliness_packets_sent;
711     RTI_ATOMIC(RTI_UINT64) manual_liveliness_packets_received;
712     RTI_ATOMIC(RTI_UINT64) endpoint_discovery_bytes_sent;
713     RTI_ATOMIC(RTI_UINT64) endpoint_discovery_bytes_received;
714     RTI_ATOMIC(RTI_UINT64) endpoint_discovery_packets_sent;
715     RTI_ATOMIC(RTI_UINT64) endpoint_discovery_packets_received;
716     /*
717      * Overhead statistics are for packets that use a builtin port (so are not
718      * user data), but are also not associated with participant or endpoint
719      * discovery
720      */
721     RTI_ATOMIC(RTI_UINT64) overhead_bytes_sent;
722     RTI_ATOMIC(RTI_UINT64) overhead_bytes_received;
723     RTI_ATOMIC(RTI_UINT64) overhead_packets_sent;
724     RTI_ATOMIC(RTI_UINT64) overhead_packets_received;
725     RTI_ATOMIC(RTI_UINT64) binding_ping_event_count;
726     RTI_ATOMIC(RTI_UINT64) binding_ping_packets_sent;
727     RTI_ATOMIC(RTI_UINT64) binding_ping_bytes_sent;
728     RTI_ATOMIC(RTI_UINT64) binding_ping_packets_received;
729     RTI_ATOMIC(RTI_UINT64) binding_ping_bytes_received;
730
731     /*
732      * Array of counters, one per DiffServ value. These counters will be
733      * incremented based on the received DiffServ values of the UDP packet.
734      *
735      * If received_priority_count[2] == 45 it means that we have received 45
736      * packets with DiffServ value 2.
737      *
738      * This field is populated only when gather_detailed_statistics is set
739      * to 1.
740      */
741     RTI_ATOMIC(RTI_UINT32)
742     received_priority_count[NDDS_TRANSPORT_UDP_DIFF_SERV_MAX_VALUE + 1];
743
744     /*****
745      * PRIVATE FIELDS

```

```

746      *****/
747      /*
748      * This field is used differently depending on whether or not we are using
749      * security. If we are not using security, this field will count the
750      * automatic liveliness messages sent and received. If we are using
751      * security, this field will count both automatic and manual liveliness
752      * messages sent and received.
753      */
754      RTI_ATOMIC(RTI_UINT64) _liveliness_bytes_sent;
755      RTI_ATOMIC(RTI_UINT64) _liveliness_bytes_received;
756      RTI_ATOMIC(RTI_UINT64) _liveliness_packets_sent;
757      RTI_ATOMIC(RTI_UINT64) _liveliness_packets_received;
758  };
759
760  /*i
761  * @brief Initializes NDDS_Transport_UDP_Statistics.
762  */
763  NDDS_Transport_DllExport void NDDS_Transport_UDP_Statistics_initialize(
764      struct NDDS_Transport_UDP_Statistics *stats);
765
766  /*i
767  * @brief Copies the content of an NDDS_Transport_UDP_Statistics object to
768  * another.
769  */
770  NDDS_Transport_DllExport void NDDS_Transport_UDP_copy_statistics(
771      struct NDDS_Transport_UDP_Statistics *dst,
772      const struct NDDS_Transport_UDP_Statistics *src);
773
774  /*i
775  * @brief Gets transport UDP statistics from self into statistics_out.
776  */
777  NDDS_Transport_DllExport void NDDS_Transport_UDP_get_statistics(
778      NDDS_Transport_Plugin *self,
779      struct NDDS_Transport_UDP_Statistics *statistics_out);
780
781  /*i
782  * @brief Gets the minimum and maximum transport priorities of the self's send
783  * resources.
784  */
785  NDDS_Transport_DllExport void NDDS_Transport_UDP_get_priority_range(
786      NDDS_Transport_Plugin *self,
787      RTI_INT32 *min_priority,
788      RTI_INT32 *max_priority);
789
790  /*i
791  * @brief Prints an address with a leading indent of "indent" spaces and a
792  * description string "%s:".
793  */
794  NDDS_Transport_DllExport void NDDS_Transport_UDP_Address_print(
795      const NDDS_Transport_Address_t *me,
796      const char *desc,
797      RTI_INT32 indent);
798
799  /*i
800  * @brief Gets bytes sent by an UDP transport in a time interval from before and
801  * after transport statistics.
802  */
803  NDDS_Transport_DllExport RTI_UINT64 NDDS_Transport_UDP_get_bytes_sent(
804      const struct NDDS_Transport_UDP_Statistics * const before,
805      const struct NDDS_Transport_UDP_Statistics * const after);
806
807  /*i
808  * @brief Gets packets sent by an UDP transport in a time interval from before
809  * and after transport statistics.
810  */
811  NDDS_Transport_DllExport RTI_UINT64 NDDS_Transport_UDP_get_packets_sent(
812      const struct NDDS_Transport_UDP_Statistics * const before,
813      const struct NDDS_Transport_UDP_Statistics * const after);
814
815  /*i
816  * @brief Gets bytes received by an UDP transport in a time interval from
817  * before and after transport statistics.
818  */
819  NDDS_Transport_DllExport RTI_UINT64 NDDS_Transport_UDP_get_bytes_received(
820      const struct NDDS_Transport_UDP_Statistics * const before,
821      const struct NDDS_Transport_UDP_Statistics * const after);
822
823  /*i
824  * @brief Gets packets received by an UDP transport in a time interval
825  * from before and after transport statistics.
826  */

```

```

827 NDDS_Transport_DllExport RTI_UINT64 NDDS_Transport_UDP_get_packets_received(
828     const struct NDDS_Transport_UDP_Statistics * const before,
829     const struct NDDS_Transport_UDP_Statistics * const after);
830
831 /* ===== */
832 /*                      UDP WAN UUID                      */
833 /* ===== */
834 /* Universal Unique IDentifier for a transport UDP_WAN instance. */
835 typedef struct {
836     RTI_UINT8 value[9];
837 } NDDS_Transport_UDP_WAN_UUID;
838
839 extern NDDS_Transport_DllVariable const NDDS_Transport_UDP_WAN_UUID
840     NDDS_Transport_UDP_WAN_UUID_ZERO;
841
842 extern NDDS_Transport_DllExport RTIBool
843 NDDS_Transport_UDP_WAN_UUID_is_zero(NDDS_Transport_UDP_WAN_UUID *uuid);
844
845 extern NDDS_Transport_DllExport RTIBool NDDS_Transport_UDP_WAN_UUID_populate(
846     NDDS_Transport_UDP_WAN_UUID *self,
847     const struct RTIOsapiRtpsGuidPrefix *guidPrefix,
848     RTI_UINT32 hostAddress);
849
850 /* ===== */
851 /*                      UDPv4 WAN Transport Address Flag    */
852 /* ===== */
853
854 typedef RTI_UINT8 NDDS_Transport_UDP_WAN_AddressFlag;
855
856
857 #define NDDS_TRANSPORT_UDP_WAN_ADDRESS_FLAG_NONE (0)
858
859 /*
860  * The U flag indicates if the locator contains a UUID. While this identifier by
861  * itself cannot be directly used to reach the transport instance, the UUID can
862  * be mapped to a public IP address by other Connex DDS applications and
863  * services. Also, a locator can have both, the U flag, and the P flag, enabled
864  * simultaneously. Locators with the U flag set are called UUID locators.
865  */
866 #define NDDS_TRANSPORT_UDP_WAN_ADDRESS_FLAG_U (0x01U << 0)
867
868 /*
869  * The P flag indicates that the locator contains a globally public IP address
870  * and public port where a transport instance can be reached. public_ip_address
871  * contains the public IP address and public_port contains the public UDP port.
872  * The port is always used to receive data and if the B flag is set it is also
873  * used to send data. Locators with the P flag set are called PUBLIC locators.
874  */
875 #define NDDS_TRANSPORT_UDP_WAN_ADDRESS_FLAG_P (0x01U << 1)
876
877 /*
878  * The B flag indicates if the locator is unidirectional or bidirectional.
879  * Bidirectional locators can send/receive RTPS traffic. Unidirectional locators
880  * can only receive RTPS traffic.
881  */
882 #define NDDS_TRANSPORT_UDP_WAN_ADDRESS_FLAG_B (0x01U << 2)
883
884 /*
885  * The R flag indicates if the locator is a relay locator provided by a Relay
886  * Service. Locators with the R flag set are called RELAY locators.
887  */
888 #define NDDS_TRANSPORT_UDP_WAN_ADDRESS_FLAG_R (0x01U << 3)
889
890 /* ===== */
891 /*                      UDPv4 WAN Transport Address Public Port */
892 /* ===== */
893
894 typedef struct {
895     RTI_UINT8 network_ordered_value[2];
896 } NDDS_Transport_UDP_WAN_NetworkPort;
897
898 #define NDDS_TRANSPORT_UDP_WAN_NETWORK_PORT_INVALID {{0}}
899
900 /* ===== */
901 /*                      UDPv4 WAN Transport Address Public Address */
902 /* ===== */
903
904 typedef struct {
905     RTI_UINT8 network_ordered_value[4];
906 } NDDS_Transport_UDP_WAN_NetworkIpv4Address;
907
908 #define NDDS_TRANSPORT_UDP_WAN_NETWORK_IP_ADDRESS_INVALID {{0}}
909
910 /* ===== */

```



```

918 /*                      UDPv4 WAN Transport Address                      */
919 /* ===== */
920 /*
921  * UDPv4_WAN locators use the following NDDS_Transport_Address_t mapping:
922  *
923  * +-----+-----+-----+-----+
924  * | Flags |                               |
925  * +-----+                               +
926  * |           DDS_Octet UUID[9]           |
927  * |           +-----+-----+           |
928  * |           | public_port |           |
929  * +-----+-----+-----+-----+
930  * | DDS_Octet public_ip_address[4] |
931  * +-----+-----+-----+-----+
932  */
933
934 typedef struct {
935     NDDS_Transport_UDP_WAN_AddressFlag _flags;
936     NDDS_Transport_UDP_WAN_UUID _uuid;
937     NDDS_Transport_UDP_WAN_NetworkPort _publicPort;
938     NDDS_Transport_UDP_WAN_NetworkIpv4Address _publicAddress;
939 } NDDS_Transport_UDPv4_WAN_Address;
940
941 extern NDDS_Transport_DllExport NDDS_Transport_UDP_Port
942 NDDS_Transport_UDPv4_WAN_Address_get_public_port(
943     const NDDS_Transport_UDPv4_WAN_Address *self);
944
945 extern NDDS_Transport_DllExport NDDS_Transport_UDP_Port
946 NDDS_Transport_UDPv4_WAN_Address_get_public_port_for_rtps_port(
947     const NDDS_Transport_UDPv4_WAN_Address *self,
948     NDDS_Transport_Port_t rtps_port,
949     RTI_INT32 udp_port_offset);
950
951 extern NDDS_Transport_DllExport void
952 NDDS_Transport_UDPv4_WAN_Address_set_public_port(
953     NDDS_Transport_UDPv4_WAN_Address *self,
954     NDDS_Transport_UDP_Port public_port);
955
956 extern NDDS_Transport_DllExport RTI_UINT32
957 NDDS_Transport_UDPv4_WAN_Address_get_public_address(
958     const NDDS_Transport_UDPv4_WAN_Address *self);
959
960 extern NDDS_Transport_DllExport void
961 NDDS_Transport_UDPv4_WAN_Address_set_public_address(
962     NDDS_Transport_UDPv4_WAN_Address *self,
963     RTI_UINT32 public_address);
964
965 extern NDDS_Transport_DllExport void NDDS_Transport_UDP_combine_addresses(
966     const NDDS_Transport_Plugin *self,
967     NDDS_Transport_Address_t *address_out,
968     const NDDS_Transport_Address_t *first_address_in,
969     const NDDS_Transport_Address_t *second_address_in);
970
971 #ifdef __cplusplus
972 } /* extern "C" */
973 #endif
974
975 #include "transport/transport_udp_impl.h"
976
977 #endif /* transport_udp_h */

```

7.7 transport_udp4.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_udp4_h
12 #define transport_udp4_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_UDP4_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_UDP4_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_UDP4_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_UDP4_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_UDP4_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_UDP4_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_UDP4_MESSAGE_SIZE_MAX_DEFAULT (9216)
293 #else
294 #define NDDS_TRANSPORT_UDP4_MESSAGE_SIZE_MAX_DEFAULT \
295     NDDS_TRANSPORT_UDP4_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_UDP4_WAN_MESSAGE_SIZE_MAX_DEFAULT (1400)
302
303 #define NDDS_TRANSPORT_UDP4_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_UDP4_MULTICAST_TTL_DEFAULT \
310     NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT
311
312
313 #define NDDS_TRANSPORT_UDP4_USE_MULTICAST_DEFAULT \
314     NDDS_TRANSPORT_UDP_USE_MULTICAST_DEFAULT
315
316
317 /*e \dref_UDPv4Transport_IGNORE_LOOPBACK_DISABLED
318 */
319 #define NDDS_TRANSPORT_UDP4_IGNORE_LOOPBACK_DISABLED \
320     NDDS_TRANSPORT_UDP_IGNORE_LOOPBACK_DISABLED
321
322 /*e \dref_UDPv4Transport_IGNORE_LOOPBACK_ENABLED
323 */
324 #define NDDS_TRANSPORT_UDP4_IGNORE_LOOPBACK_ENABLED \
325     NDDS_TRANSPORT_UDP_IGNORE_LOOPBACK_ENABLED
326
327 /*e \dref_UDPv4Transport_IGNORE_LOOPBACK_AUTO
328 */
329 #define NDDS_TRANSPORT_UDP4_IGNORE_LOOPBACK_AUTO \
330     NDDS_TRANSPORT_UDP_IGNORE_LOOPBACK_AUTO
331
332
333 /*e \dref_UDPv4Transport_USE_CHECKSUM_DISABLED
334 */
335 #define NDDS_TRANSPORT_UDP4_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         NULL, /* plugin_enabled_function_ptr */ \
414         RTI_FALSE, /* gather_detailed_statistics */ \
415         RTI_FALSE, /* drop_samples_by_guid_prefix */ \
416         RTI_OSAPI_RTPS_GUID_PREFIX_UNKNOWN /* guid_prefix_to_drop */ \
417     } \
418 }

```

```

419 /*e \ingroup NDDS_Transport_UDPv4_WAN_Plugin
420 * \brief Use this to initialize a NDDS_Transport_UDPv4_WAN_Property_t
421 * structure.
422 */
423 #define NDDS_TRANSPORT_UDPv4_WAN_PROPERTY_DEFAULT \
424 { \
425     { \
426         NDDS_TRANSPORT_CLASSID_UDPv4_WAN, \
427         NDDS_TRANSPORT_UDPv4_WAN_ADDRESS_BIT_COUNT, \
428         NDDS_TRANSPORT_UDPv4_PROPERTIES_BITMAP_DEFAULT, \
429         NDDS_TRANSPORT_UDPv4_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT, \
430         NDDS_TRANSPORT_UDPv4_WAN_MESSAGE_SIZE_MAX_DEFAULT, \
431         NULL, \
432         0, /* allow_interfaces_list */ \
433         NULL, \
434         0, /* deny_interfaces_list */ \
435         NULL, \
436         0, /* allow_multicast_interfaces_list */ \
437         NULL, \
438         0, /* deny_multicast_interfaces_list */ \
439         NDDS_TRANSPORT_UUID_UNKNOWN, /* transport_uuid */ \
440         NULL, /* domain_participant_ptr */ \
441         NULL, /* domain_participant_forwarder */ \
442         NULL, /* thread_name_prefix */ \
443         NDDS_TRANSPORT_LENGTH_UNLIMITED, /* max_interface_count */ \
444     }, \
445     NDDS_TRANSPORT_UDPv4_SEND_SOCKET_BUFFER_SIZE_DEFAULT, \
446     NDDS_TRANSPORT_UDPv4_RECV_SOCKET_BUFFER_SIZE_DEFAULT, \
447     1, /* unicast_enabled */ \
448     /* \
449     0, /* multicast_enabled */ \
450     NDDS_TRANSPORT_UDPv4_MULTICAST_TTL_DEFAULT, 0, /* multicast \
451         loopback \
452         enabled */ \
453     NDDS_TRANSPORT_UDPv4_IGNORE_LOOPBACK_AUTO, /* (auto-)ignore \
454         loopback */ \
455     1, /* ignore_nonup_interfaces */ \
456     1, /* ignore_nonrunning_interfaces */ \
457     0, /* no_zero_copy */ \
458     NDDS_TRANSPORT_UDPv4_BLOCKING_DEFAULT, 0, /* enable_v4mapped */ \
459     /* \
460     NDDS_TRANSPORT_UDPv4_USE_CHECKSUM_ENABLED, /* use_checksum */ \
461     0, 0, 0xff /* no mapping to IP_TOS by default */ \
462     1, /* send_ping */ \
463     /* \
464     0, /* don't force polling interface tracker by default */ \
465     500, /* 500 millisecs is the default polling period */ \
466     1, /* reuse multicast receive resource */ \
467     28, /* protocol_overhead_max (UDP(8)+IP(20)) */ \
468     0, /* disable interface tracking */ \
469     5000, /* join_multicast_group_timeout */ \
470     NULL, /* public_address */ \
471     NULL, 0, /* comm_ports_list */ \
472     NULL, /* wan_listener */ \
473     125, /* port_offset */ \
474     1000, /* binding_ping_period */ \
475     NULL, /* plugin_enabled_function_ptr */ \
476     RTI_FALSE, /* gather_detailed_statistics */ \
477     RTI_FALSE, /* drop_samples_by_guid_prefix */ \
478     RTI_OSAPI_RTPS_GUID_PREFIX_UNKNOWN /* guid_prefix_to_drop */ \
479 } \
480
481 /* ===== */
482 /* InterfaceListener Class */
483 /* ===== */
484
485 /*i
486 * \defgroup NDDS_Transport_UDP_InterfaceListenerClass
487 * NDDS_Transport_UDP_InterfaceListener \ingroup NDDS_Transport_UDP_Plugin
488 * \brief Interface for IPv4 interface acceptance/rejection
489 *
490 * @mtsafety See NDDS_Transport_UDP_public UML diagram
491 *
492 * @see NDDS_Transport_UDPv4_newI
493 */
494 #define NDDS_Transport_UDPv4_InterfaceListener \
495     NDDS_Transport_UDP_InterfaceListener
496
497 /*i \ingroup NDDS_Transport_UDPv4_InterfaceListenerClass
498 * \brief A user-provided function to be used to filter out particular
499 * network interfaces to be used by the Transport Plugin.

```

```

500 *
501 * The user may implement this function and pass it to an
502 * instance of an UDPv4 Transport Plugin to reject or accept network
503 * interfaces that the plugin offers as ones to use to receive
504 * messages.
505 *
506 * When \ndds contacts other applications, it will provide a list of
507 * interfaces that other applications can use to send messages (on a
508 * plugin by plugin basis). The addresses of the interfaces are used
509 * as the destination addresses of messages that are sent. If a plugin
510 * rejects one of the available network interfaces, then that interface
511 * will not be offered as one to which other plugins should send
512 * messages.
513 *
514 * For interfaces that can be used by the plugin, this function also
515 * enables the user to select which interfaces \ndds is allowed to use
516 * in sending multicast messages. The set of interfaces enabled for
517 * multicast sending is a subset of the interfaces that can be used for
518 * receiving. That is, an interface cannot be enabled to send
519 * multicast messages but disabled for receiving unicast messages.
520 *
521 * Only IP network interfaces that are "UP" (enabled by the OS and
522 * ready to use) will be passed to this filter function. The ones that
523 * are "DOWN" will not be used by definition.
524 *
525 * @param plugin \st_in The instance of the plugin for which the filter
526 * is being called.
527 *
528 * @param multicast_enabled_out \st_out The filter should set whether
529 * or not the plugin is allowed to use this interface for
530 * sending/receiving multicast messages.
531 *
532 * @param interface_in \st_in An interface offered by the plugin as a
533 * candidate for receive messages.
534 *
535 * @return 1 to use the interface, 0 to reject using the interface.
536 *         \c multicast_enabled_out will indicate whether it is also OK
537 *         to use the interface to send multicast messages.
538 */
539 #define NDDS_Transport_UDPv4_Interface_Filter_Fcn \
540     NDDS_Transport_UDP_Interface_Filter_Fcn
541
542 /* ===== */
543 /*          UDPv4/IPv4 InterfaceListener          */
544 /* ===== */
545
546
547 /*i \ingroup NDDS_Transport_UDPv4_InterfaceListenerClass
548 * \brief Default implementation of the Interface listener.
549 *
550 * The default interface listener applies the interface selection preferences
551 * specified in the NDDS_Transport_UDP_Property_t#parent:
552 * - NDDS_Transport_Property_t#allow_interfaces_list,
553 * - NDDS_Transport_Property_t#deny_interfaces_list,
554 * - NDDS_Transport_Property_t#allow_multicast_interfaces_list,
555 * - NDDS_Transport_Property_t#deny_multicast_interfaces_list
556 *
557 * The filter assumes that the strings passed in the address filter
558 * are in UDPv4 dot notation, possibly containing wildcards. For example:
559 *     10.10.30.232
560 *     10.10.*.*
561 *     192.168.1.*
562 * etc.
563 *
564 * @see NDDS_Transport_UDP_Interface_Filter_Fcn
565 */
566 extern NDDS_Transport_DllExport RTI_INT32
567 NDDS_Transport_UDPv4_InterfaceListener_onInterface(
568     struct NDDS_Transport_UDPv4_InterfaceListener *self,
569     struct NDDS_Transport_UDPv4 *plugin,
570     RTI_INT32 *multicast_enabled_out,
571     NDDS_Transport_Interface_t *interface_in,
572     const char *interface_name_in);
573
574 /*i \ingroup NDDS_Transport_UDPv4_InterfaceListenerClass
575 * \brief Default interface listener.
576 */
577 extern NDDS_Transport_DllVariable struct NDDS_Transport_UDP_InterfaceListener
578     NDDS_Transport_UDPv4_g_interfaceListener;
579
580 /* ===== */

```



```

581 /*                      Socket Factory Class                      */
582 /* ===== */
583 /*i
584 * \defgroup NDDS_Transport_UDP_SocketFactoryClass
585 * NDDS_Transport_UDP_SocketFactory \ingroup NDDS_Transport_UDPv4_Plugin \brief
586 * Interface for IPv4 socket creation and destruction.
587 *
588 * @mtsafety See NDDS_Transport_UDPv4_public UML diagram
589 *
590 * @see NDDS_Transport_UDPv4_new
591 */
592
593 #define NDDS_Transport_UDPv4_SocketFactory_Send_Socket_Create_Fcn \
594     NDDS_Transport_UDP_SocketFactory_Send_Socket_Create_Fcn
595
596
597 #define NDDS_Transport_UDPv4_SocketFactory_Receive_Socket_Create_Fcn \
598     NDDS_Transport_UDP_SocketFactory_Receive_Socket_Create_Fcn
599
600
601 #define NDDS_Transport_UDPv4_SocketFactory_Socket_Destroy_Fcn \
602     NDDS_Transport_UDP_SocketFactory_Socket_Destroy_Fcn
603
604
605 #define NDDS_Transport_UDPv4_SocketFactory NDDS_Transport_UDP_SocketFactory
606
607
608 #define NDDS_Transport_UDPv4_SocketFactory_is_valid \
609     NDDS_Transport_UDP_SocketFactory_is_valid
610
611 /* ===== */
612 /*                      IP Socket Factory Class                      */
613 /* ===== */
614
615 /*i \ingroup NDDS_Transport_UDP_SocketFactoryClass
616 * \brief Default implementation of the socket creation factory method
617 * for UDPv4 transport.
618 *
619 * @post Sending socket is not bound.
620 *
621 * @see NDDS_Transport_UDP_SocketFactory_Send_Socket_Create_Fcn
622 */
623 extern NDDS_Transport_DllExport RTI_INT32
624 NDDS_Transport_UDPv4_SocketFactory_create_send_socket (
625     struct NDDS_Transport_UDPv4_SocketFactory *me,
626     NDDS_Transport_Port_t *port_out,
627     const struct NDDS_Transport_UDPv4 *ip_in,
628     const NDDS_Transport_Address_t *interface_address_in,
629     const int *interface_index_in);
630
631 /*i \ingroup NDDS_Transport_UDP_SocketFactoryClass
632 * \brief Default implementation of the socket creation factory method.
633 *
634 * @post socket is created and bound
635 *
636 * @see NDDS_Transport_UDP_SocketFactory_Receive_Socket_Create_Fcn
637 */
638 extern NDDS_Transport_DllExport RTI_INT32
639 NDDS_Transport_UDPv4_SocketFactory_create_receive_socket (
640     struct NDDS_Transport_UDPv4_SocketFactory *me,
641     NDDS_Transport_Port_t *port_inout,
642     const struct NDDS_Transport_UDPv4 *ip_in,
643     const NDDS_Transport_Address_t *multicast_address_in);
644
645 /*i \ingroup NDDS_Transport_UDP_SocketFactoryClass
646 * \brief Default implementation of the socket destroy factory method
647 * @see NDDS_Transport_UDP_SocketFactory_Socket_Destroy_Fcn
648 */
649 extern NDDS_Transport_DllExport RTI_INT32
650 NDDS_Transport_UDPv4_SocketFactory_destroy_socket (
651     struct NDDS_Transport_UDPv4_SocketFactory *me,
652     RTI_INT32 s_in);
653
654 /*i \ingroup NDDS_Transport_UDP_SocketFactoryClass
655 * \brief Used to bind a socket to the desired port
656 *
657 * @param s_in \b In. socket to be bound
658 *
659 * @param port_desired_in \b In. port number to bind to, use 0 to be
660 * assigned an ephemeral port.
661 *

```

```

662 * @return if successful, the port number bound to or 0 if
663 * unsuccessful.
664 */
665 extern NDDS_Transport_DllExport NDDS_Transport_Port_t
666 NDDS_Transport_UDPv4_Socket_bind(
667     RTI_INT32 s_in,
668     const NDDS_Transport_Port_t port_desired_in);
669
670 /*i \ingroup NDDS_Transport_UDP_SocketFactoryClass
671 * \brief Used to bind a socket to the desired port with specified IP
672 *
673 * @param s_in \b In. socket to be bound
674 *
675 * @param port_desired_in \b In. port number to bind to, use 0 to be
676 * assigned an ephemeral port.
677 *
678 * @param ip_in \b In. IP to bind
679 *
680 * @return if successful, the port number bound to or 0 if
681 * unsuccessful.
682 */
683 extern NDDS_Transport_DllExport NDDS_Transport_Port_t
684 NDDS_Transport_UDPv4_Socket_bind_with_ip(
685     RTI_INT32 s_in,
686     const NDDS_Transport_Port_t port_desired_in,
687     unsigned int ip_in);
688
689 /*i \ingroup NDDS_Transport_UDP_SocketFactoryClass
690 * \brief Default socket factory.
691 */
692 extern NDDS_Transport_DllVariable struct NDDS_Transport_UDPv4_SocketFactory
693     NDDS_Transport_UDPv4_g_socketFactory;
694
695 /*i \ingroup NDDS_Transport_UDPv4_Plugin
696 * \brief Realization of NDDS_Transport_Get_Class_Name_Fcn_cEA for IPv4
697 * transports.
698 *
699 * Returns the string "udpv4".
700 *
701 * @see NDDS_Transport_Get_Class_Name_Fcn_cEA for complete
702 * documentation.
703 */
704 #define NDDS_Transport_UDPv4_get_class_name_cEA \
705     NDDS_Transport_UDP_get_class_name_cEA
706
707 /*e \ingroup NDDS_Transport_UDPv4_Plugin
708 * \brief Realization of NDDS_Transport_String_To_Address_Fcn_cEA for IP
709 * transports.
710 *
711 * Converts a host name string to a IPv4 address.
712 *
713 * @param self NOT USED. May be NULL.
714 *
715 * @param address_out \st_out The corresponding numerical value in IPv4
716 * format.
717 *
718 * @param address_in \st_in The name of the IPv4 address. It can be a
719 * dot notation name or a host name. If NULL, then the IP address of
720 * the localhost will be returned.
721 *
722 * @see NDDS_Transport_String_To_Address_Fcn_cEA for complete
723 * documentation.
724 */
725 #define NDDS_Transport_UDPv4_string_to_address_cEA \
726     NDDS_Transport_UDP_string_to_address_cEA
727
728 /* ===== */
729 /* UDP/IP Transport Plugin */
730 /* ===== */
731
732 /*e \ingroup NDDS_Transport_UDPv4_Plugin
733 * \brief Create an instance of a UDPv4 Transport Plugin.
734 *
735 * An application may create and register multiple instances of this
736 * Transport Plugin with \ndds. Multiple instances of the plugin may be useful
737 * for partitioning the network interfaces across multiple \ndds domains.
738 * However, note that the underlying transport, the operating system's IP layer,
739 * is still a "singleton." For example, if a unicast transport has already
740 * bound to a port, and another unicast transport tries to bind to the same
741 * port, the second attempt will fail.
742 */

```

```

743 * The transport plugin honors the interface/multicast "white" and "black"
744 * lists specified in the NDDS_Transport_UDP_Property_t#parent:
745 * - NDDS_Transport_Property_t#allow_interfaces_list,
746 * - NDDS_Transport_Property_t#deny_interfaces_list,
747 * - NDDS_Transport_Property_t#allow_multicast_interfaces_list,
748 * - NDDS_Transport_Property_t#deny_multicast_interfaces_list
749 *
750 * The format of a string in these lists is assumed to be in standard IPv4
751 * dot notation, possibly containing wildcards. For example:
752 *     - 10.10.30.232
753 *     - 10.10.*.*
754 *     - 192.168.1.*
755 *     - etc.
756 * Strings not in the correct format will be ignored.
757 *
758 * @param property_in \st_in Desired behavior of this transport. May be
759 * NULL for default property.
760 *
761 * @return A UDPv4 Transport Plugin instance on success; or NULL on failure.
762 */
763 extern NDDS_Transport_DllExport NDDS_Transport_Plugin *NDDS_Transport_UDPv4_new(
764     const struct NDDS_Transport_UDPv4_Property_t *property_in);
765
766 /*e \ingroup NDDS_Transport_UDPv4_WAN_Plugin
767 * @brief Create an instance of a UDPv4_WAN Transport Plugin.
768 *
769 * An application may create and register multiple instances of this
770 * Transport Plugin with \ndds. Multiple instances of the plugin may be useful
771 * for partitioning the network interfaces across multiple \ndds domains.
772 * However, note that the underlying transport, the operating system's IP layer,
773 * is still a "singleton." For example, if a unicast transport has already
774 * bound to a port, and another unicast transport tries to bind to the same
775 * port, the second attempt will fail.
776 *
777 * The transport plugin honors the interface "white" and "black"
778 * lists specified in the NDDS_Transport_UDP_Property_t#parent:
779 * - NDDS_Transport_Property_t#allow_interfaces_list,
780 * - NDDS_Transport_Property_t#deny_interfaces_list,
781 *
782 * The format of a string in these lists is assumed to be in standard IPv4
783 * dot notation, possibly containing wildcards. For example:
784 *     - 10.10.30.232
785 *     - 10.10.*.*
786 *     - 192.168.1.*
787 *     - etc.
788 * Strings not in the correct format will be ignored.
789 *
790 * @param property_in \st_in Desired behavior of this transport. May be
791 * NULL for default property.
792 *
793 * @return A UDPv4_WAN Transport Plugin instance on success; or NULL on failure.
794 */
795 extern NDDS_Transport_DllExport NDDS_Transport_Plugin *
796 NDDS_Transport_UDPv4_WAN_new(
797     const struct NDDS_Transport_UDPv4_WAN_Property_t *property_in);
798
799 /*i \ingroup NDDS_Transport_UDPv4_Plugin
800 * \brief Create an instance of a UDPv4 Transport Plugin.
801 *
802 * \copydoc NDDS_Transport_UDPv4_new
803 *
804 * @param socket_factory_in \st_in The factory that will create all the
805 * sockets to be used in the transport. May be NULL if default factory
806 * is to be used. Only used by RTI internal testing.
807 * End users should set this parameter to NULL.
808 *
809 * @param interface_filter_in \st_in A function that will be used to
810 * filter out network interfaces that this plugin can use. May be NULL if
811 * default interface listener is to be used. The default listener uses
812 * The NDDS_Transport_UDP_Property_t#parent's lists to select the interfaces.
813 * Only used by RTI internal testing. End users should set this
814 * parameter to NULL.
815 *
816 * @exception RTI_LOG_MALLOCFailure_d RTI_LOG_CREATION_FAILURE_s
817 * RTI_LOG_PRECONDITION_FAILURE RTI_LOG_INIT_FAILURE_s
818 */
819 extern NDDS_Transport_DllExport NDDS_Transport_Plugin *
820 NDDS_Transport_UDPv4_newI(
821     const struct NDDS_Transport_UDPv4_Property_t *property_in,
822     struct NDDS_Transport_UDPv4_SocketFactory *socket_factory_in,
823     struct NDDS_Transport_UDPv4_InterfaceListener *interface_listener_in);

```

```

824
825 /*i \ingroup NDDS_Transport_UDPv4_WAN_Plugin
826 * @brief Create an instance of a UDPv4_WAN Transport Plugin.
827 *
828 * \copydoc NDDS_Transport_UDPv4_WAN_new
829 *
830 * @param socket_factory_in \st_in The factory that will create all the
831 * sockets to be used in the transport. May be NULL if default factory
832 * is to be used. Only used by RTI internal testing.
833 * End users should set this parameter to NULL.
834 *
835 * @param interface_filter_in \st_in A function that will be used to
836 * filter out network interfaces that this plugin can use. May be NULL if
837 * default interface listener is to be used. The default listener uses
838 * The NDDS_Transport_UDP_Property_t#parent's lists to select the interfaces.
839 * Only used by RTI internal testing. End users should set this
840 * parameter to NULL.
841 *
842 * @exception RTI_LOG_MALLOC_FAILURE_d RTI_LOG_CREATION_FAILURE_s
843 * RTI_LOG_PRECONDITION_FAILURE RTI_LOG_INIT_FAILURE_s
844 */
845 extern NDDS_Transport_DllExport NDDS_Transport_Plugin *
846 NDDS_Transport_UDPv4_WAN_newI(
847     const struct NDDS_Transport_UDPv4_WAN_Property_t *property_in,
848     struct NDDS_Transport_UDPv4_SocketFactory *socket_factory_in,
849     struct NDDS_Transport_UDPv4_InterfaceListener *interface_listener_in);
850
851
852 #define NDDS_Transport_UDPv4_get_receive_interfaces_cEA \
853     NDDS_Transport_UDP_get_receive_interfaces_cEA
854
855
856 #define NDDS_Transport_UDPv4_get_ignore_loopback_interface_status \
857     NDDS_Transport_UDP_get_ignore_loopback_interface_status
858
859
860 #define NDDS_Transport_UDPv4_is_local_address \
861     NDDS_Transport_UDP_is_local_address
862
863
864 #define NDDS_Transport_UDPv4_map_transport_priority \
865     NDDS_Transport_UDP_map_transport_priority
866
867 /*i \ingroup NDDS_Transport_UDPv4_Plugin
868 * \brief Definition of IPv4 local host.
869 */
870 #define NDDS_TRANSPORT_UDPv4_ADDRESS_LOCALHOST \
871     { \
872     { \
873         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 127, 0, 0, 1 \
874     } \
875     }
876
877
878 #define NDDS_TRANSPORT_UDPv4_UNBLOCK_MESSAGE_SIZE \
879     NDDS_TRANSPORT_UDP_UNBLOCK_MESSAGE_SIZE
880
881
882 #define NDDS_Transport_UDPv4_Statistics NDDS_Transport_UDP_Statistics
883
884
885 #define NDDS_Transport_UDPv4_get_statistics NDDS_Transport_UDP_get_statistics
886
887
888 #define NDDS_Transport_UDPv4_get_priority_range \
889     NDDS_Transport_UDP_get_priority_range
890
891 #define NDDS_Transport_UDPv4_Address_print NDDS_Transport_UDP_Address_print
892
893 extern NDDS_Transport_DllExport RTIBool
894 NDDS_Transport_UDPv4_WAN_Address_to_string(
895     const NDDS_Transport_Address_t *me,
896     char *buffer,
897     size_t buffer_size_in);
898
899 extern NDDS_Transport_DllExport void NDDS_Transport_UDPv4_WAN_Address_print(
900     const NDDS_Transport_Address_t *me,
901     const char *desc,
902     RTI_INT32 indent);
903
904

```

```

905 #ifdef __cplusplus
906 } /* extern "C" */
907 #endif
908
909 #include "transport/transport_udp4_impl.h"
910
911 #endif /* transport_udp4_h */

```

7.8 transport_udp6.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_udp6_h
12 #define transport_udp6_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     /** \dref_UDPv6Transport_Property_t_ignore_loopback_interface */
71     RTI_INT32 ignore_loopback_interface;
72
73     /** \dref_UDPv6Transport_Property_t_ignore_nonrunning_interfaces */
74     RTI_INT32 ignore_nonrunning_interfaces;
75
76     /** \dref_UDPv6Transport_Property_t_no_zero_copy */
77     RTI_INT32 no_zero_copy;
78
79     /** \dref_UDPv6Transport_Property_t_send_blocking */
80     RTI_INT32 send_blocking;
81
82     /** \dref_UDPv6Transport_Property_t_enable_v4mapped */
83     RTI_INT32 enable_v4mapped;
84
85     /** \dref_UDPv6Transport_Property_t_transport_priority_mask */
86     RTI_UINT32 transport_priority_mask;
87
88     /** \dref_UDPv6Transport_Property_t_transport_priority_mapping_low */
89     RTI_INT32 transport_priority_mapping_low;
90
91     /** \dref_UDPv6Transport_Property_t_transport_priority_mapping_high */
92     RTI_INT32 transport_priority_mapping_high;
93
94     /** \dref_UDPv6Transport_Property_t_send_ping */
95     RTI_INT32 send_ping;
96
97     /** \dref_UDPv6Transport_Property_t_force_interface_poll_detection */
98     RTI_INT32 force_interface_poll_detection;
99
100    /** \dref_UDPv6Transport_Property_t_interface_poll_period */
101    RTI_UINT32 interface_poll_period;
102
103    /** \dref_UDPv6Transport_Property_t_reuse_multicast_receive_resource */
104    RTI_INT32 reuse_multicast_receive_resource;
105
106    /** \dref_UDPv6Transport_Property_t_protocol_overhead_max */
107    RTI_INT32 protocol_overhead_max;
108
109    /** \dref_UDPv6Transport_Property_t_disable_interface_tracking */
110    RTI_INT32 disable_interface_tracking;
111
112    /** \dref_UDPv6Transport_Property_t_join_multicast_group_timeout */
113    RTI_UINT32 join_multicast_group_timeout;
114
115    /** \dref_UDPv6Transport_Property_t_public_address */
116    char *public_address;
117
118    /** \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_UDP6_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_UDP6_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_UDP6_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_UDP6_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_UDP6_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_UDP6_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_UDP6_MESSAGE_SIZE_MAX_DEFAULT (9196)
190 #else
191     #define NDDS_TRANSPORT_UDP6_MESSAGE_SIZE_MAX_DEFAULT \
192         NDDS_TRANSPORT_UDP6_PAYLOAD_SIZE_MAX
193 #endif
194
195 #define NDDS_TRANSPORT_UDP6_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_UDP6_MULTICAST_TTL_DEFAULT \
202     NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT
203
204 #define NDDS_TRANSPORT_UDP6_USE_MULTICAST_DEFAULT \
205     NDDS_TRANSPORT_UDP_USE_MULTICAST_DEFAULT
206
207
208 /*e \dref_UDPv6Transport_IGNORE_LOOPBACK_DISABLED
209 */
210 #define NDDS_TRANSPORT_UDP6_IGNORE_LOOPBACK_DISABLED \
211     NDDS_TRANSPORT_UDP_IGNORE_LOOPBACK_DISABLED
212
213 /*e \dref_UDPv6Transport_IGNORE_LOOPBACK_ENABLED
214 */
215 #define NDDS_TRANSPORT_UDP6_IGNORE_LOOPBACK_ENABLED \
216     NDDS_TRANSPORT_UDP_IGNORE_LOOPBACK_ENABLED
217
218 /*e \dref_UDPv6Transport_IGNORE_LOOPBACK_AUTO
219 */
220 #define NDDS_TRANSPORT_UDP6_IGNORE_LOOPBACK_AUTO \
221     NDDS_TRANSPORT_UDP_IGNORE_LOOPBACK_AUTO
222
223
224 /*e \dref_UDPv6Transport_BLOCKING_NEVER
225 */
226 #define NDDS_TRANSPORT_UDP6_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         NULL, /* plugin_enabled_function_ptr */ \
293         RTI_FALSE, /* gather_detailed_statistics */ \
294         RTI_FALSE, /* drop_samples_by_guid_prefix */ \
295         RTI_OSAPI_RTPS_GUID_PREFIX_UNKNOWN /* guid_prefix_to_drop */ \
296     } \
297
298 /* ===== */
299 /* InterfaceListener Class */
300 /* ===== */
301
302 #define NDDS_Transport_UDPv6_InterfaceListener \
303     NDDS_Transport_UDP_InterfaceListener
304
305
306 #define NDDS_Transport_UDPv6_Interface_Filter_Fcn \
307     NDDS_Transport_UDP_Interface_Filter_Fcn
308
309 /* ===== */
310 /* UDP/IPv6 InterfaceListener */

```



```

311 /* ===== */
312
313
314 /*i \ingroup NDDS_Transport_UDPv6_InterfaceListenerClass
315 * \brief Default implementation of the Interface listener.
316 *
317 * The default interface listener applies the interface selection preferences
318 * specified in the NDDS_Transport_UDPv6_Property_t#parent:
319 * - NDDS_Transport_Property_t#allow_interfaces_list,
320 * - NDDS_Transport_Property_t#deny_interfaces_list,
321 * - NDDS_Transport_Property_t#allow_multicast_interfaces_list,
322 * - NDDS_Transport_Property_t#deny_multicast_interfaces_list
323 *
324 * The filter assumes that the strings passed in the address filter
325 * are in UDPv6 dot notation, possibly containing wildcards. For example:
326 * ffce:2222:::0000
327 * 0000:::0001
328 * 0*1 etc.
329 *
330 * 0::192.168.1.* (TODO: this form is not yet supported)
331 * etc.
332 *
333 * @see NDDS_Transport_UDPv6_Interface_Filter_Fcn
334 */
335 extern NDDS_Transport_DllExport RTI_INT32
336 NDDS_Transport_UDPv6_InterfaceListener_onInterface(
337     struct NDDS_Transport_UDPv6_InterfaceListener *self,
338     struct NDDS_Transport_UDPv6 *plugin,
339     RTI_INT32 *multicast_enabled_out,
340     NDDS_Transport_Interface_t *interface_in,
341     const char *interface_name_in);
342
343 /*i \ingroup NDDS_Transport_UDPv6_InterfaceListenerClass
344 * \brief Default interface listener.
345 */
346 extern NDDS_Transport_DllVariable struct NDDS_Transport_UDP_InterfaceListener
347     NDDS_Transport_UDPv6_g_interfaceListener;
348
349 /* ===== */
350 /* Socket Factory Class */
351 /* ===== */
352 /*i
353 * \defgroup NDDS_Transport_UDPv6_SocketFactoryClass
354 * NDDS_Transport_UDPv6_SocketFactory \ingroup NDDS_Transport_UDPv6_Plugin
355 * \brief Interface for IPv6 socket creation and destruction.
356 *
357 * @mtsafety See NDDS_Transport_UDPv6_public UML diagram
358 *
359 * @see NDDS_Transport_UDPv6_new
360 */
361
362 #define NDDS_Transport_UDPv6_SocketFactory_Send_Socket_Create_Fcn \
363     NDDS_Transport_UDP_SocketFactory_Send_Socket_Create_Fcn
364
365 #define NDDS_Transport_UDPv6_SocketFactory_Receive_Socket_Create_Fcn \
366     NDDS_Transport_UDP_SocketFactory_Receive_Socket_Create_Fcn
367
368 #define NDDS_Transport_UDPv6_SocketFactory_Socket_Destroy_Fcn \
369     NDDS_Transport_UDP_SocketFactory_Socket_Destroy_Fcn
370
371 #define NDDS_Transport_UDPv6_SocketFactory NDDS_Transport_UDP_SocketFactory
372
373 #define NDDS_Transport_UDPv6_SocketFactory_is_valid \
374     NDDS_Transport_UDP_SocketFactory_is_valid
375
376 /* ===== */
377 /* IP Socket Factory Class */
378 /* ===== */
379
380 /*i \ingroup NDDS_Transport_UDPv6_SocketFactoryClass
381 * \brief Default implementation of the socket creation factory method
382 * for UDPv6 transport.
383 *
384 * @post Sending socket is not bound.
385 *
386 * @see NDDS_Transport_UDPv6_SocketFactory_Send_Socket_Create_Fcn
387 */
388 extern NDDS_Transport_DllExport RTI_INT32
389 NDDS_Transport_UDPv6_SocketFactory_create_send_socket(
390     struct NDDS_Transport_UDPv6_SocketFactory *me,
391     NDDS_Transport_Port_t *port_out,

```

```

392     const struct NDDS_Transport_UDPv6 *ip_in,
393     const NDDS_Transport_Address_t *interface_address_in,
394     const int *interface_index_in);
395
396 /*i \ingroup NDDS_Transport_UDPv6_SocketFactoryClass
397 * \brief Default implementation of the socket creation factory method.
398 *
399 * @post socket is created and bound
400 *
401 * @see NDDS_Transport_UDPv6_SocketFactory_Receive_Socket_Create_Fcn
402 */
403 extern NDDS_Transport_DllExport RTI_INT32
404 NDDS_Transport_UDPv6_SocketFactory_create_receive_socket (
405     struct NDDS_Transport_UDPv6_SocketFactory *me,
406     NDDS_Transport_Port_t *port_inout,
407     const struct NDDS_Transport_UDPv6 *ip_in,
408     const NDDS_Transport_Address_t *multicast_address_in);
409
410 /*i \ingroup NDDS_Transport_UDPv6_SocketFactoryClass
411 * \brief Default implementation of the socket destroy factory method
412 * @see NDDS_Transport_UDPv6_SocketFactory_Socket_Destroy_Fcn
413 */
414 extern NDDS_Transport_DllExport RTI_INT32
415 NDDS_Transport_UDPv6_SocketFactory_destroy_socket (
416     struct NDDS_Transport_UDPv6_SocketFactory *me,
417     RTI_INT32 s_in);
418
419 /*i \ingroup NDDS_Transport_UDPv6_SocketFactoryClass
420 * \brief Used to bind a socket to the desired port
421 *
422 * @param s_in \b In. socket to be bound
423 *
424 * @param port_desired_in \b In. port number to bind to, use 0 to be
425 * assigned an ephemeral port.
426 *
427 * @return if successful, the port number bound to or 0 if
428 * unsuccessful.
429 */
430 extern NDDS_Transport_DllExport NDDS_Transport_Port_t
431 NDDS_Transport_UDPv6_Socket_bind(
432     RTI_INT32 s_in,
433     const NDDS_Transport_Port_t port_desired_in);
434
435 /*i \ingroup NDDS_Transport_UDPv6_SocketFactoryClass
436 * \brief Used to bind a socket to the desired port with specified IP
437 *
438 * @param s_in \b In. socket to be bound
439 *
440 * @param port_desired_in \b In. port number to bind to, use 0 to be
441 * assigned an ephemeral port.
442 *
443 * @param ip_in \b In. IP to bind
444 *
445 * @return if successful, the port number bound to or 0 if
446 * unsuccessful.
447 */
448 extern NDDS_Transport_DllExport NDDS_Transport_Port_t
449 NDDS_Transport_UDPv6_Socket_bind_with_ip(
450     RTI_INT32 s_in,
451     const NDDS_Transport_Port_t port_desired_in,
452     const struct in6_addr *ip_in);
453
454 /*i \ingroup NDDS_Transport_UDPv6_SocketFactoryClass
455 * \brief Default socket factory.
456 */
457 extern NDDS_Transport_DllVariable struct NDDS_Transport_UDPv6_SocketFactory
458     NDDS_Transport_UDPv6_g_socketFactory;
459
460 #define NDDS_Transport_UDPv6_get_class_name_cEA \
461     NDDS_Transport_UDP_get_class_name_cEA
462
463 /*e \ingroup NDDS_Transport_UDPv6_Plugin
464 * \brief Realization of NDDS_Transport_String_To_Address_Fcn_cEA for IP
465 * transports.
466 *
467 * Converts a host name string to a IPv6 address.
468 *
469 * @param self NOT USED. May be NULL.
470 *
471 * @param address_out \st_out The corresponding numerical value in IPv6
472 * format.

```

```

473 *
474 * @param address_in \st_in The name of the IPv6 address. It can be a
475 * dot notation name or a host name. If NULL, then the IP address of
476 * the localhost will be returned.
477 *
478 * @see NDDS_Transport_String_To_Address_Fcn_cEA for complete
479 * documentation.
480 */
481 #define NDDS_Transport_UDPv6_string_to_address_cEA \
482     NDDS_Transport_UDP_string_to_address_cEA
483
484 /* ===== */
485 /*             UDP/IPv6 Transport Plugin             */
486 /* ===== */
487
488 /*e \ingroup NDDS_Transport_UDPv6_Plugin
489 * \brief Create an instance of a UDPv6 Transport Plugin.
490 *
491 * An application may create and register multiple instances of this
492 * Transport Plugin with \ndds. Multiple instances of the plugin may be useful
493 * for partitioning the network interfaces across multiple \ndds domains.
494 * However, note that the underlying transport, the operating system's IP layer,
495 * is still a "singleton." For example, if a unicast transport has already
496 * bound to a port, and another unicast transport tries to bind to the same
497 * port, the second attempt will fail.
498 *
499 * The transport plugin honors the interface/multicast "white" and "black"
500 * lists specified in the NDDS_Transport_UDPv6_Property_t#parent:
501 * - NDDS_Transport_Property_t#allow_interfaces_list,
502 * - NDDS_Transport_Property_t#deny_interfaces_list,
503 * - NDDS_Transport_Property_t#allow_multicast_interfaces_list,
504 * - NDDS_Transport_Property_t#deny_multicast_interfaces_list
505 *
506 * The format of a string in these lists is assumed to be in standard IPv6
507 * dot notation, possibly containing wildcards. For example:
508 * - 10.10.30.232
509 * - 10.10.*.*
510 * - 192.168.1.*
511 * - etc.
512 * Strings not in the correct format will be ignored.
513 *
514 * @param property_in \st_in Desired behavior of this transport. May be
515 * NULL for default property.
516 *
517 * @return A UDPv6 Transport Plugin instance on success; or NULL on failure.
518 */
519 extern NDDS_Transport_DllExport NDDS_Transport_Plugin *NDDS_Transport_UDPv6_new(
520     const struct NDDS_Transport_UDPv6_Property_t *property_in);
521
522 /*i \ingroup NDDS_Transport_UDPv6_Plugin
523 * \brief Create an instance of a UDPv6 Transport Plugin.
524 *
525 * \copydoc NDDS_Transport_UDPv6_new
526 *
527 * @param socket_factory_in \st_in The factory that will create all the
528 * sockets to be used in the transport. May be NULL if default factory
529 * is to be used. Only used by RTI internal testing.
530 * End users should set this parameter to NULL.
531 *
532 * @param interface_filter_in \st_in A function that will be used to
533 * filter out network interfaces that this plugin can use. May be NULL if
534 * default interface listener is to be used. The default listener uses
535 * The NDDS_Transport_UDPv6_Property_t#parent's lists to select the interfaces.
536 * Only used by RTI internal testing. End users should set this
537 * parameter to NULL.
538 *
539 * @exception RTI_LOG_MALLOC_FAILURE_d RTI_LOG_CREATION_FAILURE_s
540 * RTI_LOG_PRECONDITION_FAILURE RTI_LOG_INIT_FAILURE_s
541 */
542 extern NDDS_Transport_DllExport NDDS_Transport_Plugin *
543 NDDS_Transport_UDPv6_newI(
544     const struct NDDS_Transport_UDPv6_Property_t *property_in,
545     struct NDDS_Transport_UDPv6_SocketFactory *socket_factory_in,
546     struct NDDS_Transport_UDPv6_InterfaceListener *interface_listener_in);
547
548 #define NDDS_Transport_UDPv6_get_receive_interfaces_cEA \
549     NDDS_Transport_UDP_get_receive_interfaces_cEA
550
551 #define NDDS_Transport_UDPv6_get_ignore_loopback_interface_status \
552     NDDS_Transport_UDP_get_ignore_loopback_interface_status
553

```

```
554 #define NDDS_Transport_UDPv6_is_local_address \
555     NDDS_Transport_UDP_is_local_address
556
557 #define NDDS_Transport_UDPv6_map_transport_priority \
558     NDDS_Transport_UDP_map_transport_priority
559
560 /*i \ingroup NDDS_Transport_UDPv6_Plugin
561  * \brief Definition of local host.
562  */
563 #define NDDS_TRANSPORT_UDPv6_ADDRESS_LOCALHOST \
564     { \
565         { \
566             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 \
567         } \
568     }
569
570 #define NDDS_TRANSPORT_UDPv6_UNBLOCK_MESSAGE_SIZE \
571     NDDS_TRANSPORT_UDP_UNBLOCK_MESSAGE_SIZE
572
573 #define NDDS_Transport_UDPv6_Statistics NDDS_Transport_UDP_Statistics
574
575 #define NDDS_Transport_UDPv6_get_statistics NDDS_Transport_UDP_get_statistics
576
577 #define NDDS_Transport_UDPv6_get_priority_range \
578     NDDS_Transport_UDP_get_priority_range
579
580 #define NDDS_Transport_UDPv6_Address_print NDDS_Transport_UDP_Address_print
581
582
583 #ifdef __cplusplus
584 } /* extern "C" */
585 #endif
586
587 #include "transport/transport_udp6_impl.h"
588
589 #endif /* transport_udp6_h */
```

Chapter 8

Example Documentation

8.1 HelloWorld.idl

8.1.1 IDL Type Description

The data type to be disseminated by RTI Connex 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. 2109) = HelloWorld, for use in the application code.

- HelloWorld.h
- HelloWorld.c

User Data Type Support (p. 210) types as required by the DDS specification for use in the application code.

- HelloWorldSupport.h
- HelloWorldSupport.c

Methods required by the RTI Connex implementation. These contains the auto-generated methods for serializing and deserializing the type.

- HelloWorldPlugin.h
- HelloWorldPlugin.c

8.1.1.1 HelloWorld.idl

```
struct HelloWorld {  
    string<128> msg;  
};
```

8.2 HelloWorld.c

8.2.1 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. 2109) = HelloWorld
- **FooSeq** (p. 2113) = HelloWorldSeq

8.2.1.1 HelloWorld.h

```

/*
WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

This file was generated from HelloWorld.idl
using RTI Code Generator (rtiddsgen) version 4.6.0.
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_c_h
#include "ndds/ndds_c.h"
#endif
#include "cdr/cdr_typeCode.h"
#else
#include "ndds_standalone_type.h"
#endif
#ifdef __cplusplus
extern "C" {
    #endif
    extern const char *HelloWorldTYPENAME;
    typedef struct HelloWorld
    {
        DDS_Char * msg;
    } HelloWorld ;
    #if defined(NDDS_USER_DLL_EXPORT) && defined(RTI_WIN32)
    #undef NDDUSERDllExport
    #define NDDUSERDllExport __declspec(dllexport)
    #endif
    #if !defined(RTI_WIN32) && defined(NDDS_USER_SYMBOL_EXPORT)
    #undef NDDUSERDllExport
    #define NDDUSERDllExport __attribute__((visibility("default")))
    #endif
    #ifndef NDDS_STANDALONE_TYPE
    NDDUSERDllExport DDS_TypeCode * HelloWorld_get_typecode(void); /* Type code */
    NDDUSERDllExport RTIXCdrTypePlugin *HelloWorld_get_type_plugin_info(void);
    NDDUSERDllExport RTIXCdrSampleAccessInfo *HelloWorld_get_sample_access_info(void);
    #endif
    DDS_SEQUENCE(HelloWorldSeq, HelloWorld);
    NDDUSERDllExport
    RTIBool HelloWorld_initialize(
        HelloWorld* self);
    NDDUSERDllExport
    RTIBool HelloWorld_initialize_ex(
        HelloWorld* self, RTIBool allocatePointers, RTIBool allocateMemory);
    NDDUSERDllExport
    RTIBool HelloWorld_initialize_w_params(
        HelloWorld* self,
        const struct DDS_TypeAllocationParams_t * allocParams);
    NDDUSERDllExport
    RTIBool HelloWorld_finalize_w_return(
        HelloWorld* self);
    NDDUSERDllExport
    void HelloWorld_finalize(
        HelloWorld* self);
    NDDUSERDllExport

```

```

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
#ifdef __cplusplus
}
#endif
#endif /* HelloWorld */

```

8.2.1.2 HelloWorld.c

```

/*
WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

This file was generated from HelloWorld.idl
using RTI Code Generator (rtiddsgen) version 4.6.0.
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_c_h
#include "ndds/ndds_c.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
#ifndef osapi_heap_h
#include "osapi/osapi_heap.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
/* ===== */
const char *HelloWorld_TYPENAME = "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 */
    RTIXCdrTypeCodeAnnotations_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 = (RTIXCdrTypeCode *)&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;
}
RTIXCdrSampleAccessInfo *HelloWorld_get_sample_access_info()
{
    static RTI_ATOMIC(RTIBool) is_initialized;
    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;
    }
    HelloWorld_g_memberAccessInfos[0].bindingMemberValueOffset[0] =
    offsetof(struct HelloWorld, msg);
    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.languageBinding =
    RTI_XCDR_TYPE_BINDING_C ;
    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;
        }
        if (deallocParams) {} /* To avoid warnings */
        deallocParamsTmp.delete_pointers = (DDS_Boolean)deletePointers;
        deallocParamsTmp.delete_optional_members = DDS_BOOLEAN_TRUE;
    }
    RTIBool HelloWorld_copy(
        HelloWorld* dst,
        const HelloWorld* src)
    {
        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;
    }
}
#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"
#else
#include "dds_c_sequence_TSeq.gen"
#endif
#undef T_copy
#undef T_finalize_w_params

```

```
#undef T_initialize_w_params
#undef TSeq
#undef T
```

8.3 HelloWorldSupport.c

8.3.1 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. 210), where:

- **FooTypeSupport** (p. 2114) = HelloWorldTypeSupport
- **FooDataWriter** (p. 2113) = HelloWorldDataWriter
- **FooDataReader** (p. 2112) = HelloWorldDataReader

8.3.1.1 HelloWorldSupport.h

```
/*
WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

This file was generated from HelloWorld.idl
using RTI Code Generator (rtiddsgen) version 4.6.0.
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"
#ifdef ndds_c_h
#include "ndds/ndds_c.h"
#endif
#ifdef __cplusplus
extern "C" {
    #endif
    #if (defined(RTI_WIN32) || defined(RTI_WINCE) || defined(RTI_INTIME)) && defined(NDDS_USER_DLL_EXPORT)
    #endif
    /* ===== */
    #if defined(NDDS_USER_DLL_EXPORT) && defined(RTI_WIN32)
    #undef NDDUSERDllExport
    #define NDDUSERDllExport __declspec(dllexport)
    #endif
    #if !defined(RTI_WIN32) && defined(NDDS_USER_SYMBOL_EXPORT)
    #undef NDDUSERDllExport
    #define NDDUSERDllExport __attribute__((visibility("default")))
    #endif
    DDS_TYPESUPPORT_C(HelloWorldTypeSupport, HelloWorld);
    DDS_DATAWRITER_WITH_DATA_CONSTRUCTOR_METHODS_C(HelloWorldDataWriter, HelloWorld);
    DDS_DATAREADER_C(HelloWorldDataReader, HelloWorldSeq, HelloWorld);
    #if defined(NDDS_USER_DLL_EXPORT) || defined(NDDS_USER_SYMBOL_EXPORT)
    #undef NDDUSERDllExport
    #define NDDUSERDllExport
    #endif
    #ifdef __cplusplus
}
#endif
#endif /* HelloWorldSupport_1436885487_h */
```

8.3.1.2 HelloWorldSupport.c

```

/*
WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

This file was generated from HelloWorld.idl
using RTI Code Generator (rtiddsgen) version 4.6.0.
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"
/* ===== */
/* ----- */
/* DDSDataWriter
*/
/* Requires */
#define TYPENAME HelloWorldTYPENAME
/* Defines */
#define TDataWriter HelloWorldDataWriter
#define TData HelloWorld
#define RTI_ENABLE_TDATAWRITER_DATA_CONSTRUCTOR_METHODS
#include "dds_c/generic/dds_c_data_TDataWriter.gen"
#undef RTI_ENABLE_TDATAWRITER_DATA_CONSTRUCTOR_METHODS
#undef TDataWriter
#undef TData
#undef TYPENAME
/* ----- */
/* DDSDataReader
*/
/* Requires */
#define TYPENAME HelloWorldTYPENAME
/* Defines */
#define TDataReader HelloWorldDataReader
#define TDataSeq HelloWorldSeq
#define TData HelloWorld
#define RTI_ENABLE_TDATAREADER_DATA_CONSISTENCY_CHECK_METHOD
#include "dds_c/generic/dds_c_data_TDataReader.gen"
#undef RTI_ENABLE_TDATAREADER_DATA_CONSISTENCY_CHECK_METHOD
#undef TDataReader
#undef TDataSeq
#undef TData
#undef TYPENAME
/* ----- */
/* TypeSupport

«IMPLEMENTATION »

Requires: TYPENAME,
TPlugin_new
TPlugin_delete
Defines: TTypeSupport, TData, TDataReader, TDataWriter
*/
/* Requires */
#define TYPENAME 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_c/generic/dds_c_data_TTypeSupport.gen"
#undef TTypeSupport
#undef TData
#undef TDataReader
#undef TDataWriter
#ifndef NDDS_STANDALONE_TYPE
#undef TGENERATE_TYPECODE
#endif
#undef TGENERATE_SER_CODE
#undef TYPENAME
#undef TPlugin_new
#undef TPlugin_delete

```

8.4 HelloWorldPlugin.c

8.4.1 RTI Connex 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 Connex implementation.

8.4.1.1 HelloWorldPlugin.h

```

/*
WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

This file was generated from HelloWorld.idl
using RTI Code Generator (rtiddsgen) version 4.6.0.
The rtiddsgen tool is part of the RTI Connex DDS distribution.
For more information, type 'rtiddsgen -help' at a command shell
or consult the Code Generator User's Manual.
*/
#ifdef HelloWorldPlugin_1436885487_h
#define HelloWorldPlugin_1436885487_h
#include "HelloWorld.h"
struct RTICdrStream;
#ifdef pres_typePlugin_h
#include "pres/pres_typePlugin.h"
#endif
#if defined(NDDS_USER_DLL_EXPORT) && defined(RTI_WIN32)
#undef NDDUSUSERDllExport
#define NDDUSUSERDllExport __declspec(dllexport)
#else
#if !defined(RTI_WIN32) && defined(NDDS_USER_SYMBOL_EXPORT)
#undef NDDUSUSERDllExport
#define NDDUSUSERDllExport __attribute__((visibility("default")))
#endif
#ifdef __cplusplus
extern "C" {
#endif
#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:
* ----- */
NDDUSUSERDllExport extern HelloWorld*
HelloWorldPluginSupport_create_data_w_params(
    const struct DDS_TypeAllocationParams_t * alloc_params);
NDDUSUSERDllExport extern HelloWorld*
HelloWorldPluginSupport_create_data_ex(RTIBool allocate_pointers);
NDDUSUSERDllExport extern HelloWorld*
HelloWorldPluginSupport_create_data(void);
NDDUSUSERDllExport extern RTIBool
HelloWorldPluginSupport_copy_data(
    HelloWorld *out,
    const HelloWorld *in);
NDDUSUSERDllExport extern void
HelloWorldPluginSupport_destroy_data_w_params(
    HelloWorld *sample,
    const struct DDS_TypeDeallocationParams_t * dealloc_params);
NDDUSUSERDllExport extern void
HelloWorldPluginSupport_destroy_data_ex(
    HelloWorld *sample, RTIBool deallocate_pointers);
NDDUSUSERDllExport extern void
HelloWorldPluginSupport_destroy_data(
    HelloWorld *sample);
NDDUSUSERDllExport 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);
NDDUSUSERDllExport extern
struct RTIXCdrInterpreterPrograms * HelloWorldPlugin_get_programs(void);
/* Plugin Functions */
NDDUSUSERDllExport extern struct PRESTypePlugin*
HelloWorldPlugin_new(void);
NDDUSUSERDllExport extern void
HelloWorldPlugin_delete(struct PRESTypePlugin *);
#ifdef __cplusplus
}
#endif
#if defined(NDDUS_USER_DLL_EXPORT) || defined(NDDUS_USER_SYMBOL_EXPORT)
#undef NDDUSUSERDllExport
#define NDDUSUSERDllExport
#endif
#endif /* HelloWorldPlugin_1436885487_h */

```

8.4.1.2 HelloWorldPlugin.c

```

/*
WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

This file was generated from HelloWorld.idl
using RTI Code Generator (rtiddsgen) version 4.6.0.
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>
#ifdef ndds_c_h
#include "ndds/ndds_c.h"
#endif
#ifdef osapi_type_h
#include "osapi/osapi_type.h"
#endif
#ifdef osapi_heap_h
#include "osapi/osapi_heap.h"
#endif
#ifdef osapi_utility_h
#include "osapi/osapi_utility.h"
#endif
#ifdef cdr_type_h
#include "cdr/cdr_type.h"
#endif
#ifdef cdr_type_object_h
#include "cdr/cdr_typeObject.h"
#endif
#ifdef cdr_encapsulation_h
#include "cdr/cdr_encapsulation.h"
#endif
#ifdef cdr_stream_h
#include "cdr/cdr_stream.h"
#endif
#include "xcdr/xcdr_interpreter.h"
#include "xcdr/xcdr_stream.h"
#ifdef cdr_log_h
#include "cdr/cdr_log.h"
#endif
#ifdef 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 "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;
    }
    RTIOsapiHeap_allocateStructure(&(sample), 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] */
        RTIOsapiHeap_freeStructure(sample);
        sample=NULL;
    }
    return sample;
}

HelloWorld *
HelloWorldPluginSupport_create_data_ex(RTIBool allocate_pointers)
{
    HelloWorld *sample = NULL;
    RTIOsapiHeap_allocateStructure(&(sample), 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] */
        RTIOsapiHeap_freeStructure(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);
    RTIOsapiHeap_freeStructure(sample);
}

```



```

void
HelloWorldPluginSupport_destroy_data_ex(
    HelloWorld *sample, RTIBool deallocate_pointers) {
    HelloWorld_finalize_ex(sample, deallocate_pointers);
    RTIOsapiHeap_freeStructure(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;
    if (registration_data) {} /* To avoid warnings */
    if (participant_info) {} /* To avoid warnings */
    if (top_level_registration) {} /* To avoid warnings */
    if (container_plugin_context) {} /* To avoid warnings */
    if (type_code) {} /* To avoid warnings */
    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;
    if (top_level_registration) {} /* To avoid warnings */
    if (containerPluginContext) {} /* To avoid warnings */
    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)
{
    if (endpoint_data) {} /* To avoid warnings */
    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_INITIALIZER;
    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_INITIALIZER;
    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;
    if (drop_sample) {} /* To avoid warnings */
    /* Depending on the type and the flags used in rttiddsgen, 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)
{
    struct RTIXCdrInterpreterProgramsGenProperty programProperty =
        RTIXCdrInterpreterProgramsGenProperty_INITIALIZER;
    struct RTIXCdrInterpreterPrograms *retPrograms = NULL;
    programProperty.generateWithOnlyKeyFields = RTI_XCDR_FALSE;
    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;
    retPrograms =
        DDS_TypeCodeFactory_assert_programs_in_global_list(
            DDS_TypeCodeFactory_get_instance(),
            HelloWorld_get_typecode(),
            &programProperty,
            RTI_XCDR_SER_PROGRAM
            | RTI_XCDR_DESER_PROGRAM
            | RTI_XCDR_GET_MAX_SER_SIZE_PROGRAM
            | RTI_XCDR_GET_SER_SIZE_PROGRAM);
    return retPrograms;
}
/* -----

```

```

* 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_TYPE_PLUGIN_DDS_TYPE;
    /* 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;
    return plugin;
}
void
HelloWorldPlugin_delete(struct PRESTypePlugin *plugin)
{
    RTIOsapiHeap_freeStructure(plugin);
}
#undef RTI_CDR_CURRENT_SUBMODULE

```

8.5 HelloWorld_publisher.c

8.5.1 RTI Connexx 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.

8.5.1.1 HelloWorld_publisher.c

```

/*
 * (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 Connexx 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.
 */
/* HelloWorld_publisher.c

A publication of data of type HelloWorld

This file is derived from code automatically generated by the rtiddsgen
command:

rtiddsgen -language C -example <arch> HelloWorld.idl

Example publication of type HelloWorld automatically generated by
'rtiddsgen'. To test it, follow these steps:

(1) Compile this file and the example subscription.

(2) Start the subscription on the same domain used for RTI Connexx

(3) Start the publication on the same domain used for RTI Connexx

(4) [Optional] Specify the list of discovery initial peers and
multicast receive addresses via an environment variable or a file
(in the current working directory) called NDDS_DISCOVERY_PEERS.

You can run any number of publisher and subscriber programs, and can
add and remove them dynamically from the domain.
*/
#include <stdio.h>

```



```

#include <stdlib.h>
#include "ndds/ndds_c.h"
#include "HelloWorld.h"
#include "HelloWorldSupport.h"
/* Delete all entities */
static int publisher_shutdown(
    DDS_DomainParticipant *participant)
{
    DDS_ReturnCode_t retcode;
    int status = 0;
    if (participant != NULL) {
        retcode = DDS_DomainParticipant_delete_contained_entities(participant);
        if (retcode != DDS_RETCODE_OK) {
            fprintf(stderr, "delete_contained_entities error %d\n", retcode);
            status = -1;
        }
        retcode = DDS_DomainParticipantFactory_delete_participant(
            DDS_TheParticipantFactory, participant);
        if (retcode != DDS_RETCODE_OK) {
            fprintf(stderr, "delete_participant error %d\n", retcode);
            status = -1;
        }
    }
    /* RTI Data Distribution Service provides finalize_instance() method on
    domain participant factory for people who want to release memory used
    by the participant factory. Uncomment the following block of code for
    clean destruction of the singleton. */
    /*
    retcode = DDS_DomainParticipantFactory_finalize_instance();
    if (retcode != DDS_RETCODE_OK) {
        fprintf(stderr, "finalize_instance error %d\n", retcode);
        status = -1;
    }
    */
    return status;
}

int publisher_main(int domainId, int sample_count)
{
    DDS_DomainParticipant *participant = NULL;
    DDS_Publisher *publisher = NULL;
    DDS_Topic *topic = NULL;
    DDS_DataWriter *writer = NULL;
    HelloWorldDataWriter *HelloWorld_writer = NULL;
    HelloWorld *instance = NULL;
    DDS_ReturnCode_t retcode;
    DDS_InstanceHandle_t instance_handle = DDS_HANDLE_NIL;
    const char *type_name = NULL;
    int count = 0;
    struct DDS_Duration_t send_period = {4,0};
    /* To customize participant QoS, use
    the configuration file USER_QOS_PROFILES.xml */
    participant = DDS_DomainParticipantFactory_create_participant(
        DDS_TheParticipantFactory, domainId, &DDS_PARTICIPANT_QOS_DEFAULT,
        NULL /* listener */, DDS_STATUS_MASK_NONE);
    if (participant == NULL) {
        fprintf(stderr, "create_participant error\n");
        publisher_shutdown(participant);
        return -1;
    }
    /* To customize publisher QoS, use
    the configuration file USER_QOS_PROFILES.xml */
    publisher = DDS_DomainParticipant_create_publisher(
        participant, &DDS_PUBLISHER_QOS_DEFAULT, NULL /* listener */,
        DDS_STATUS_MASK_NONE);
    if (publisher == NULL) {
        fprintf(stderr, "create_publisher error\n");
        publisher_shutdown(participant);
        return -1;
    }
    /* Register type before creating topic */
    type_name = HelloWorldTypeSupport_get_type_name();
    retcode = HelloWorldTypeSupport_register_type(
        participant, type_name);
    if (retcode != DDS_RETCODE_OK) {
        fprintf(stderr, "register_type error %d\n", retcode);
        publisher_shutdown(participant);
        return -1;
    }
    /* To customize topic QoS, use
    the configuration file USER_QOS_PROFILES.xml */
    topic = DDS_DomainParticipant_create_topic(

```

```

    participant, "Example HelloWorld",
    type_name, &DDS_TOPIC_QOS_DEFAULT, NULL /* listener */,
    DDS_STATUS_MASK_NONE);
if (topic == NULL) {
    fprintf(stderr, "create_topic error\n");
    publisher_shutdown(participant);
    return -1;
}
/* To customize data writer QoS, use
the configuration file USER_QOS_PROFILES.xml */
writer = DDS_Publisher_create_datawriter(
    publisher, topic,
    &DDS_DATAWRITER_QOS_DEFAULT, NULL /* listener */, DDS_STATUS_MASK_NONE);
if (writer == NULL) {
    fprintf(stderr, "create_datawriter error\n");
    publisher_shutdown(participant);
    return -1;
}
HelloWorld_writer = HelloWorldDataWriter_narrow(writer);
if (HelloWorld_writer == NULL) {
    fprintf(stderr, "DataWriter narrow error\n");
    publisher_shutdown(participant);
    return -1;
}
/* Create data sample for writing */
instance = HelloWorldTypeSupport_create_data_ex(DDS_BOOLEAN_TRUE);
if (instance == NULL) {
    fprintf(stderr, "HelloWorldTypeSupport_create_data error\n");
    publisher_shutdown(participant);
    return -1;
}
/* For a data type that has a key, if the same instance is going to be
written multiple times, initialize the key here
and register the keyed instance prior to writing */
/*
instance_handle = HelloWorldDataWriter_register_instance(
    HelloWorld_writer, instance);
*/
/* Main loop */
for (count=0; (sample_count == 0) || (count < sample_count); ++count) {
    printf("Writing HelloWorld, count %d\n", count);
    /* Modify the data to be written here */
    /* Write data */
    retcode = HelloWorldDataWriter_write(
        HelloWorld_writer, instance, &instance_handle);
    if (retcode != DDS_RETCODE_OK) {
        fprintf(stderr, "write error %d\n", retcode);
    }
    NDDS_Utility_sleep(&send_period);
}
/*
retcode = HelloWorldDataWriter_unregister_instance(
    HelloWorld_writer, instance, &instance_handle);
if (retcode != DDS_RETCODE_OK) {
    fprintf(stderr, "unregister instance error %d\n", retcode);
}
*/
/* Delete data sample */
retcode = HelloWorldTypeSupport_delete_data_ex(instance, DDS_BOOLEAN_TRUE);
if (retcode != DDS_RETCODE_OK) {
    fprintf(stderr, "HelloWorldTypeSupport_delete_data error %d\n", retcode);
}
/* Cleanup and delete delete all entities */
return publisher_shutdown(participant);
}
int main(int argc, char *argv[])
{
    int domain_id = 0;
    int sample_count = 0; /* infinite loop */
    if (argc >= 2) {
        domain_id = atoi(argv[1]);
    }
    if (argc >= 3) {
        sample_count = atoi(argv[2]);
    }
    /* Uncomment this to turn on additional logging
    NDDS_Config_Logger_set_verbosity_by_category(
        NDDS_Config_Logger_get_instance(),
        NDDS_CONFIG_LOG_CATEGORY_API,
        NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL);
    */

```

```

    return publisher_main(domain_id, sample_count);
}

```

8.6 HelloWorld_subscriber.c

8.6.1 RTI Connex Subscription Example

The unmodified subscription example generated by rtiddsgen (see the Code Generator User's Manual for more information).

8.6.1.1 HelloWorld_subscriber.c

```

/*
 * (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 Connex 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.
 */
/* HelloWorld_subscriber.c

```

A subscription example

This file is derived from code automatically generated by the rtiddsgen command:

```
rtiddsgen -language C -example <arch> HelloWorld.idl
```

Example subscription of type HelloWorld automatically generated by 'rtiddsgen'. To test it, follow these steps:

- (1) Compile this file and the example publication.
- (2) Start the subscription on the same domain used for RTI Connex
- (3) Start the publication on the same domain used for RTI Connex
- (4) [Optional] Specify the list of discovery initial peers and multicast receive addresses via an environment variable or a file (in the current working directory) called NDDS_DISCOVERY_PEERS.

You can run any number of publisher and subscriber programs, and can add and remove them dynamically from the domain.

```

*/
#include <stdio.h>
#include <stdlib.h>
#include "ndds/ndds_c.h"
#include "HelloWorld.h"
#include "HelloWorldSupport.h"
void HelloWorldListener_on_requested_deadline_missed(
    void* listener_data,
    DDS_DataReader* reader,
    const struct DDS_RequestedDeadlineMissedStatus *status)
{
}
void HelloWorldListener_on_requested_incompatible_qos(
    void* listener_data,
    DDS_DataReader* reader,
    const struct DDS_RequestedIncompatibleQosStatus *status)
{
}
void HelloWorldListener_on_sample_rejected(
    void* listener_data,
    DDS_DataReader* reader,
    const struct DDS_SampleRejectedStatus *status)
{
}

```

```

}
void HelloWorldListener_on_liveliness_changed(
    void* listener_data,
    DDS_DataReader* reader,
    const struct DDS_LivelinessChangedStatus *status)
{
}
void HelloWorldListener_on_sample_lost(
    void* listener_data,
    DDS_DataReader* reader,
    const struct DDS_SampleLostStatus *status)
{
}
void HelloWorldListener_on_subscription_matched(
    void* listener_data,
    DDS_DataReader* reader,
    const struct DDS_SubscriptionMatchedException *status)
{
}
void HelloWorldListener_on_data_available(
    void* listener_data,
    DDS_DataReader* reader)
{
    HelloWorldDataReader *HelloWorld_reader = NULL;
    struct HelloWorldSeq data_seq = DDS_SEQUENCE_INITIALIZER;
    struct DDS_SampleInfoSeq info_seq = DDS_SEQUENCE_INITIALIZER;
    DDS_ReturnCode_t retcode;
    int i;
    HelloWorld_reader = HelloWorldDataReader_narrow(reader);
    if (HelloWorld_reader == NULL) {
        fprintf(stderr, "DataReader narrow error\n");
        return;
    }
    retcode = HelloWorldDataReader_take(
        HelloWorld_reader,
        &data_seq, &info_seq, DDS_LENGTH_UNLIMITED,
        DDS_ANY_SAMPLE_STATE, DDS_ANY_VIEW_STATE, DDS_ANY_INSTANCE_STATE);
    if (retcode == DDS_RETCODE_NO_DATA) {
        return;
    }
    else if (retcode != DDS_RETCODE_OK) {
        fprintf(stderr, "take error %d\n", retcode);
        return;
    }
    for (i = 0; i < HelloWorldSeq_get_length(&data_seq); ++i) {
        if (DDS_SampleInfoSeq_get_reference(&info_seq, i) -> valid_data) {
            printf("Received data\n");
            HelloWorldTypeSupport_print_data(
                HelloWorldSeq_get_reference(&data_seq, i));
        }
    }
    retcode = HelloWorldDataReader_return_loan(
        HelloWorld_reader,
        &data_seq, &info_seq);
    if (retcode != DDS_RETCODE_OK) {
        fprintf(stderr, "return loan error %d\n", retcode);
    }
}
/* Delete all entities */
static int subscriber_shutdown(
    DDS_DomainParticipant *participant)
{
    DDS_ReturnCode_t retcode;
    int status = 0;
    if (participant != NULL) {
        retcode = DDS_DomainParticipant_delete_contained_entities(participant);
        if (retcode != DDS_RETCODE_OK) {
            fprintf(stderr, "delete_contained_entities error %d\n", retcode);
            status = -1;
        }
        retcode = DDS_DomainParticipantFactory_delete_participant(
            DDS_TheParticipantFactory, participant);
        if (retcode != DDS_RETCODE_OK) {
            fprintf(stderr, "delete_participant error %d\n", retcode);
            status = -1;
        }
    }
}
/* RTI Data Distribution Service provides the finalize_instance() method on
domain participant factory for users who want to release memory used
by the participant factory. Uncomment the following block of code for
clean destruction of the singleton. */
/*

```

```

    retcode = DDS_DomainParticipantFactory_finalize_instance();
    if (retcode != DDS_RETCODE_OK) {
        fprintf(stderr, "finalize_instance error %d\n", retcode);
        status = -1;
    }
    */
    return status;
}
int subscriber_main(int domainId, int sample_count)
{
    DDS_DomainParticipant *participant = NULL;
    DDS_Subscriber *subscriber = NULL;
    DDS_Topic *topic = NULL;
    struct DDS_DataReaderListener reader_listener =
        DDS_DataReaderListener_INITIALIZER;
    DDS_DataReader *reader = NULL;
    DDS_ReturnCode_t retcode;
    const char *type_name = NULL;
    int count = 0;
    struct DDS_Duration_t poll_period = {4,0};
    /* To customize participant QoS, use
    the configuration file USER_QOS_PROFILES.xml */
    participant = DDS_DomainParticipantFactory_create_participant(
        DDS_TheParticipantFactory, domainId, &DDS_PARTICIPANT_QOS_DEFAULT,
        NULL /* listener */, DDS_STATUS_MASK_NONE);
    if (participant == NULL) {
        fprintf(stderr, "create_participant error\n");
        subscriber_shutdown(participant);
        return -1;
    }
    /* To customize subscriber QoS, use
    the configuration file USER_QOS_PROFILES.xml */
    subscriber = DDS_DomainParticipant_create_subscriber(
        participant, &DDS_SUBSCRIBER_QOS_DEFAULT, NULL /* listener */,
        DDS_STATUS_MASK_NONE);
    if (subscriber == NULL) {
        fprintf(stderr, "create_subscriber error\n");
        subscriber_shutdown(participant);
        return -1;
    }
    /* Register the type before creating the topic */
    type_name = HelloWorldTypeSupport_get_type_name();
    retcode = HelloWorldTypeSupport_register_type(participant, type_name);
    if (retcode != DDS_RETCODE_OK) {
        fprintf(stderr, "register_type error %d\n", retcode);
        subscriber_shutdown(participant);
        return -1;
    }
    /* To customize topic QoS, use
    the configuration file USER_QOS_PROFILES.xml */
    topic = DDS_DomainParticipant_create_topic(
        participant, "Example HelloWorld",
        type_name, &DDS_TOPIC_QOS_DEFAULT, NULL /* listener */,
        DDS_STATUS_MASK_NONE);
    if (topic == NULL) {
        fprintf(stderr, "create_topic error\n");
        subscriber_shutdown(participant);
        return -1;
    }
    /* Set up a data reader listener */
    reader_listener.on_requested_deadline_missed =
        HelloWorldListener_on_requested_deadline_missed;
    reader_listener.on_requested_incompatible_qos =
        HelloWorldListener_on_requested_incompatible_qos;
    reader_listener.on_sample_rejected =
        HelloWorldListener_on_sample_rejected;
    reader_listener.on_liveliness_changed =
        HelloWorldListener_on_liveliness_changed;
    reader_listener.on_sample_lost =
        HelloWorldListener_on_sample_lost;
    reader_listener.on_subscription_matched =
        HelloWorldListener_on_subscription_matched;
    reader_listener.on_data_available =
        HelloWorldListener_on_data_available;
    /* To customize data reader QoS, use
    the configuration file USER_QOS_PROFILES.xml */
    reader = DDS_Subscriber_create_datareader(
        subscriber, DDS_Topic_as_topicdescription(topic),
        &DDS_DATAREADER_QOS_DEFAULT, &reader_listener, DDS_STATUS_MASK_ALL);
    if (reader == NULL) {
        fprintf(stderr, "create_datareader error\n");

```

```
        subscriber_shutdown(participant);
        return -1;
    }
    /* Main loop */
    for (count=0; (sample_count == 0) || (count < sample_count); ++count) {
        printf("HelloWorld subscriber sleeping for %d sec...\n",
            poll_period.sec);
        NDDS_Utility_sleep(&poll_period);
    }
    /* Cleanup and delete all entities */
    return subscriber_shutdown(participant);
}

int main(int argc, char *argv[])
{
    int domain_id = 0;
    int sample_count = 0; /* infinite loop */
    if (argc >= 2) {
        domain_id = atoi(argv[1]);
    }
    if (argc >= 3) {
        sample_count = atoi(argv[2]);
    }
    /* Uncomment this to turn on additional logging
    NDDS_Config_Logger_set_verbosity_by_category(
        NDDS_Config_Logger_get_instance(),
        NDDS_CONFIG_LOG_CATEGORY_API,
        NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL);
    */
    return subscriber_main(domain_id, sample_count);
}
```

Index

absolute_generation_rank	DDS_TransportMulticastMapping_t, 2054
DDS_SampleInfo, 1991	alive_count
accept_unknown_peers	DDS_LivelinessChangedStatus, 1821
DDS_DiscoveryQosPolicy, 1718	alive_count_change
access	DDS_LivelinessChangedStatus, 1821
DDS_ValueMember, 2089	alive_instance_count
access_scope	DDS_DataReaderCacheStatus, 1590
DDS_PresentationQosPolicy, 1893	DDS_DataWriterCacheStatus, 1643
acknowledgment_kind	alive_instance_count_peak
DDS_ReliabilityQosPolicy, 1939	DDS_DataReaderCacheStatus, 1590
active_count	DDS_DataWriterCacheStatus, 1643
DDS_ReliableReaderActivityChangedStatus, 1941	alive_instance_removal
active_count_change	DDS_DataReaderResourceLimitsInstanceReplacementSettings,
DDS_ReliableReaderActivityChangedStatus, 1942	1618
Activity Context, 1458	All QoS Policies, 1238
NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_DOMAIN_ID, 1462	NDDS_ASYNCHRONOUSPUBLISHER_QOS_POLICY_ID,
NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_ENTITY_KIND, 1462	1248
NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_ENTITY_NAME, 1462	DDS_DURABILITY_QOS_POLICY_ID, 1248
NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_GUID_PREFIX, 1461	DDS_BATCH_QOS_POLICY_ID, 1248
NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_ALL, 1460	DDS_DATA_REPRESENTATION_QOS_POLICY_ID,
NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_DEFAULT, 1460	1247
NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_NONE, 1460	DDS_DATABASE_QOS_POLICY_ID, 1248
NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_TOPIC_ID, 1461	DDS_DATAREADERPROTOCOL_QOS_POLICY_ID,
NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_TYPEDDDS, 1461	1248
NDDS_Config_ActivityContext_set_attribute_mask, 1462	DDS_DATAREADERRESOURCELIMITS_QOS_POLICY_ID,
NDDS_Config_ActivityContextAttributeKind, 1460	1247
NDDS_Config_ActivityContextAttributeKindMask, 1460	DDS_DATATAG_QOS_POLICY_ID, 1247
Additional RTI Connexx Communication Patterns, 1017	DDS_DATAWRITERPROTOCOL_QOS_POLICY_ID,
address	1248
DDS_Locator_t, 1827	DDS_DATAWRITERRESOURCELIMITS_QOS_POLICY_ID,
NDDS_Transport_Interface_t, 2121	1249
address_bit_count	DDS_DEADLINE_QOS_POLICY_ID, 1246
NDDS_Transport_Property_t, 2123	DDS_DESTINATIONORDER_QOS_POLICY_ID,
addresses	1247
	DDS_DISCOVERY_QOS_POLICY_ID, 1247
	DDS_DISCOVERYCONFIG_QOS_POLICY_ID,
	1248
	DDS_DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_ID,
	1248
	DDS_DURABILITY_QOS_POLICY_ID, 1246
	DDS_DURABILITYSERVICE_QOS_POLICY_ID,
	1247
	DDS_ENTITYFACTORY_QOS_POLICY_ID, 1247

- DDS_ENTITYNAME_QOS_POLICY_ID, 1248
- DDS_EVENT_QOS_POLICY_ID, 1248
- DDS_GROUPDATA_QOS_POLICY_ID, 1247
- DDS_HISTORY_QOS_POLICY_ID, 1247
- DDS_INVALID_QOS_POLICY_ID, 1246
- DDS_LATENCYBUDGET_QOS_POLICY_ID, 1246
- DDS_LIFESPAN_QOS_POLICY_ID, 1247
- DDS_LIVELINESS_QOS_POLICY_ID, 1247
- DDS_LOCATORFILTER_QOS_POLICY_ID, 1248
- DDS_LOGGING_QOS_POLICY_ID, 1249
- DDS_MONITORING_QOS_POLICY_ID, 1249
- DDS_MULTICHANNEL_QOS_POLICY_ID, 1248
- DDS_OWNERSHIP_QOS_POLICY_ID, 1246
- DDS_OWNERSHIPSTRENGTH_QOS_POLICY_ID, 1247
- DDS_PARTITION_QOS_POLICY_ID, 1247
- DDS_PRESENTATION_QOS_POLICY_ID, 1246
- DDS_PROFILE_QOS_POLICY_ID, 1248
- DDS_PROPERTY_QOS_POLICY_ID, 1248
- DDS_PUBLISHMODE_QOS_POLICY_ID, 1248
- DDS_QOS_POLICY_COUNT, 1246
- DDS_QosPolicyId_t, 1246
- DDS_QosPrintFormat_INITIALIZER, 1245
- DDS_READERDATA_LIFECYCLE_QOS_POLICY_ID, 1247
- DDS_RECEIVERPOOL_QOS_POLICY_ID, 1248
- DDS_RELIABILITY_QOS_POLICY_ID, 1247
- DDS_RESOURCELIMITS_QOS_POLICY_ID, 1247
- DDS_SYSTEMRESOURCELIMITS_QOS_POLICY_ID, 1248
- DDS_TIMEBASEDFILTER_QOS_POLICY_ID, 1247
- DDS_TOPICDATA_QOS_POLICY_ID, 1247
- DDS_TOPICQUERYDISPATCH_QOS_POLICY_ID, 1249
- DDS_TRANSPORTBUILTIN_QOS_POLICY_ID, 1248
- DDS_TRANSPORTMULTICAST_QOS_POLICY_ID, 1248
- DDS_TRANSPORTMULTICASTMAPPING_QOS_POLICY_ID, 1248
- DDS_TRANSPORTPRIORITY_QOS_POLICY_ID, 1247
- DDS_TRANSPORTSELECTION_QOS_POLICY_ID, 1248
- DDS_TRANSPORTUNICAST_QOS_POLICY_ID, 1248
- DDS_TYPE_CONSISTENCY_ENFORCEMENT_QOS_POLICY_ID, 1247
- DDS_TYPESUPPORT_QOS_POLICY_ID, 1248
- DDS_USERDATA_QOS_POLICY_ID, 1246
- DDS_WIREPROTOCOL_QOS_POLICY_ID, 1247
- DDS_WRITERDATA_LIFECYCLE_QOS_POLICY_ID, 1247
- allocate_optional_members
- DDS_TypeAllocationParams_t, 2070
- allocate_pointers
 - DDS_TypeAllocationParams_t, 2070
- allow_interfaces_list
 - NDDS_Transport_Property_t, 2125
- allow_interfaces_list_length
 - NDDS_Transport_Property_t, 2125
- allow_multicast_interfaces_list
 - NDDS_Transport_Property_t, 2126
- allow_multicast_interfaces_list_length
 - NDDS_Transport_Property_t, 2127
- app_ack_period
 - DDS_RtpsReliableReaderProtocol_t, 1958
- Application Acknowledgement, 835
- Application Startup, 838
- Application-Defined Metadata, 836
- application_name
 - DDS_MonitoringQosPolicy, 1854
- Applied Per-Instance, 956
- as_datareaderlistener
 - DDS_SubscriberListener, 2010
- as_datawriterlistener
 - DDS_PublisherListener, 1918
- as_listener
 - DDS_DataReaderListener, 1594
 - DDS_DataWriterListener, 1645
 - DDS_DomainParticipantListener, 1724
 - DDS_TopicListener, 2042
- as_publisherlistener
 - DDS_DomainParticipantListener, 1724
- as_readconditionparams
 - DDS_QueryConditionParams, 1928
- as_subscriberlistener
 - DDS_DomainParticipantListener, 1725
- as_topiclistener
 - DDS_DomainParticipantListener, 1724
- assertions_per_lease_duration
 - DDS_LivelinessQosPolicy, 1826
- asynchronous_batch_thread
 - DDS_AsynchronousPublisherQosPolicy, 1541
- ASYNCHRONOUS_PUBLISHER, 1250
 - DDS_ASYNCHRONOUSPUBLISHER_QOS_POLICY_NAME, 1250
- asynchronous_publisher
 - DDS_DiscoveryConfigQosPolicy, 1704
 - DDS_PublisherQos, 1919
- AsyncWaitSet, 1067
 - DDS_ASYNC_WAITSET_COMPLETION_TOKEN_IGNORE, 1090
 - DDS_ASYNC_WAITSET_COMPLETION_TOKEN_USE_IMPLICIT_ANY, 1090
 - DDS_ASYNC_WAITSET_PROPERTY_DEFAULT, 1090
 - DDS_AsyncWaitSet, 1070

- DDS_AsyncWaitSet_attach_condition, 1078
- DDS_AsyncWaitSet_attach_condition_with_completion_token, 1079
- DDS_AsyncWaitSet_create_completion_token, 1085
- DDS_AsyncWaitSet_delete, 1088
- DDS_AsyncWaitSet_delete_completion_token, 1085
- DDS_AsyncWaitSet_detach_condition, 1077
- DDS_AsyncWaitSet_detach_condition_with_completion_token, 1077
- DDS_AsyncWaitSet_get_conditions, 1080
- DDS_AsyncWaitSet_get_property, 1076
- DDS_AsyncWaitSet_is_started, 1083
- DDS_AsyncWaitSet_new, 1086
- DDS_AsyncWaitSet_new_with_listener, 1086
- DDS_AsyncWaitSet_new_with_thread_factory, 1088
- DDS_AsyncWaitSet_start, 1082
- DDS_AsyncWaitSet_start_with_completion_token, 1082
- DDS_AsyncWaitSet_stop, 1081
- DDS_AsyncWaitSet_stop_with_completion_token, 1081
- DDS_AsyncWaitSet_unlock_condition, 1080
- DDS_AsyncWaitSetCompletionToken, 1073
- DDS_AsyncWaitSetCompletionToken_wait, 1076
- DDS_AsyncWaitSetListener_INITIALIZER, 1070
- DDS_AsyncWaitSetListener_OnThreadDeletedCallback, 1074
- DDS_AsyncWaitSetListener_OnThreadSpawnedCallback, 1074
- DDS_AsyncWaitSetListener_OnWaitTimeoutCallback, 1075
- DDS_DataReaderStatusConditionHandler, 1075
- DDS_DataReaderStatusConditionHandler_delete, 1089
- DDS_DataReaderStatusConditionHandler_new, 1089
- autodispose_unregistered_instances
 - DDS_WriterDataLifecycleQosPolicy, 2107
- autoenable_created_entities
 - DDS_EntityFactoryQosPolicy, 1789
- autopurge_disposed_instances_delay
 - DDS_ReaderDataLifecycleQosPolicy, 1932
 - DDS_WriterDataLifecycleQosPolicy, 2108
- autopurge_disposed_samples_delay
 - DDS_ReaderDataLifecycleQosPolicy, 1932
- autopurge_nowriter_instances_delay
 - DDS_ReaderDataLifecycleQosPolicy, 1933
- autopurge_nowriter_samples_delay
 - DDS_ReaderDataLifecycleQosPolicy, 1932
- autopurge_remote_not_alive_writer_delay
 - DDS_DataReaderResourceLimitsQosPolicy, 1636
- autopurge_unregistered_instances_delay
 - DDS_WriterDataLifecycleQosPolicy, 2107
- autoregister_instances
 - DDS_DataWriterResourceLimitsQosPolicy, 1678
- AVAILABILITY, 1250
 - DDS_AVAILABILITY_QOS_POLICY_NAME, 1251
- availability
 - DDS_DataReaderQos, 1617
 - DDS_DataWriterQos, 1672
- available_builtin_endpoints_ext
 - DDS_ParticipantBuiltinTopicData, 1872
- Basic QoS, 783
- BATCH, 1251
 - DDS_BATCH_QOS_POLICY_NAME, 1252
- batch
 - DDS_DataWriterQos, 1672
- Batching, 838
- binding_ping_period
 - NDDS_Transport_UDPv4_WAN_Property_t, 2153
- bitmask
 - DDS_EndpointTrustProtectionInfo, 1787
 - DDS_ParticipantTrustProtectionInfo, 1880
- bits
 - DDS_StructMember, 2008
 - DDS_ValueMember, 2089
- buffer_alignment
 - DDS_ReceiverPoolQosPolicy, 1936
- buffer_initial_size
 - DDS_DynamicDataProperty_t, 1780
- buffer_max_size
 - DDS_DynamicDataProperty_t, 1780
- buffer_size
 - DDS_ReceiverPoolQosPolicy, 1935
- build
 - NDDS_Config_LibraryVersion_t, 2115
- Built-in Sequences, 718
- Built-in Topic's Trust Types, 309
 - DDS_EndpointTrustAlgorithmInfo, 312
 - DDS_EndpointTrustAttributesMask, 310
 - DDS_EndpointTrustInterceptorAlgorithmInfo, 312
 - DDS_EndpointTrustProtectionInfo, 311
 - DDS_ParticipantTrustAlgorithmInfo, 312
 - DDS_ParticipantTrustAttributesMask, 310
 - DDS_ParticipantTrustInterceptorAlgorithmInfo, 312
 - DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo, 312
 - DDS_ParticipantTrustProtectionInfo, 310
 - DDS_ParticipantTrustSignatureAlgorithmInfo, 311
 - DDS_PluginEndpointTrustAttributesMask, 311
 - DDS_PluginParticipantTrustAttributesMask, 310
 - DDS_TrustAlgorithmBit, 311
 - DDS_TrustAlgorithmRequirements, 311
 - DDS_TrustAlgorithmSet, 311
 - DDS_VendorEndpointTrustAttributesMask, 311
- Built-in Topics, 165
- Built-in Transport Plugins, 736

Built-in Types, 305

Builtin Qos Profiles, 1394

DDS_BUILTIN_QOS_LIB, 1400

DDS_BUILTIN_QOS_LIB_EXP, 1404

DDS_BUILTIN_QOS_SNIPPET_LIB, 1415

DDS_PROFILE_BASELINE, 1400

DDS_PROFILE_BASELINE_5_0_0, 1400

DDS_PROFILE_BASELINE_5_1_0, 1400

DDS_PROFILE_BASELINE_5_2_0, 1401

DDS_PROFILE_BASELINE_5_3_0, 1401

DDS_PROFILE_BASELINE_6_0_0, 1401

DDS_PROFILE_BASELINE_6_1_0, 1401

DDS_PROFILE_BASELINE_7_0_0, 1401

DDS_PROFILE_BASELINE_7_1_0, 1402

DDS_PROFILE_BASELINE_ROOT, 1400

DDS_PROFILE_GENERIC_510_TRANSPORT_COMPATIBILITY, 1405

1403

DDS_PROFILE_GENERIC_AUTO_TUNING, 1409

DDS_PROFILE_GENERIC_BEST_EFFORT, 1404

DDS_PROFILE_GENERIC_COMMON, 1402

DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY, 1414

1402

DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_FAST_FLOW, 1403

1403

DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY_MEDIUM_FLOW, 1403

1403

DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE,

1404

DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_FAST_FLOW,

1407

DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_MEDIUM_FLOW,

1408

DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA_SLOW_FLOW,

1408

DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_PERSISTENT,

1408

DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_PERSISTENT_FAST_FLOW,

1409

DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT,

1409

DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT_LOCAL,

1409

DDS_PROFILE_GENERIC_MINIMAL_MEMORY_FOOTPRINT, 1426

1410

DDS_PROFILE_GENERIC_MONITORING2, 1410

DDS_PROFILE_GENERIC_MONITORING2_WAN,

1411

DDS_PROFILE_GENERIC_MONITORING_COMMON,

1402

DDS_PROFILE_GENERIC_OTHER_DDS_VENDOR_COMPATIBILITY, 1403

1403

DDS_PROFILE_GENERIC_PARTICIPANT_LARGE_DATA,

1405

DDS_PROFILE_GENERIC_PARTICIPANT_LARGE_DATA_MONITORING,

1406

DDS_PROFILE_GENERIC_SECURITY, 1410

DDS_PROFILE_GENERIC_STRICT_RELIABLE,

1404

DDS_PROFILE_GENERIC_STRICT_RELIABLE_HIGH_THROUGHPUT,

1405

DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA,

1406

DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_FAST_FLOW,

1407

DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_MEDIUM_FLOW,

1407

DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_SLOW_FLOW,

1408

DDS_PROFILE_GENERIC_STRICT_RELIABLE_LOW_LATENCY,

1408

DDS_PROFILE_PATTERN_ALARM_EVENT, 1413

DDS_PROFILE_PATTERN_ALARM_STATUS, 1414

DDS_PROFILE_PATTERN_EVENT, 1413

DDS_PROFILE_PATTERN_LAST_VALUE_CACHE,

1414

DDS_PROFILE_PATTERN_PERIODIC_DATA, 1412

DDS_PROFILE_PATTERN_RELIABLE_STREAMING,

1413

DDS_PROFILE_PATTERN_RPC, 1412

DDS_PROFILE_PATTERN_STATUS, 1414

DDS_PROFILE_PATTERN_STREAMING, 1412

DDS_SNIPPET_5_1_0_TRANSPORT_ENABLE,

1413

DDS_SNIPPET_COMPATIBILITY_CONNEXT_MICRO_VERSION_2_4_0,

1413

DDS_SNIPPET_COMPATIBILITY_OTHER_DDS_VENDORS_ENABLE,

1413

DDS_SNIPPET_FEATURE_AUTO_TUNING_ENABLE,

1413

DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_209MBPS,

1413

DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_52MBPS,

1413

DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_838MBPS,

1413

DDS_SNIPPET_FEATURE_MONITORING2_ENABLE,

1413

DDS_SNIPPET_FEATURE_MONITORING_ENABLE,

1425

DDS_SNIPPET_FEATURE_SECURITY_ENABLE,

1426

DDS_SNIPPET_FEATURE_TOPIC_QUERY_ENABLE,

1426

DDS_SNIPPET_FEATURE_TOPIC_QUERY_ENABLE_DYNAMIC,

1418

DDS_SNIPPET_OPTIMIZATION_DISCOVERY_COMMON,

1418

DDS_SNIPPET_OPTIMIZATION_DISCOVERY_ENDPOINT_FAST,

- 1419 builtin_unicast_port_offset
- DDS_SNIPPET_OPTIMIZATION_DISCOVERY_PARTICIPANT_RESOURCE_LIMITS, 1982
- 1419 bytes_per_token
- DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_COMMON, 1802
- 1415 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_COMMON_ControllerTokenBucketProperty_t, 1802
- 1416 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_COMMON_HIGH_RATE, DDS_LoadingQosPolicy, 1832
- 1416 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_COMMON_KEEP_ALL, DDS_TypeSupportQosPolicy, 2082
- 1415 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_COMMON_KEEP_EXPRESSION, DDS_ChannelFilterExpression_max_length
- 1416 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_COMMON_KEEP_EXPRESSION, DDS_DomainParticipantResourceLimitsQosPolicy, 1752
- 1417 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LARGE_DATA, DDS_DomainParticipantResourceLimitsQosPolicy, 1752
- 1417 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LARGE_DATA, DDS_DomainParticipantResourceLimitsQosPolicy, 1752
- 1417 DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LARGE_DATA, DDS_DomainParticipantResourceLimitsQosPolicy, 1752
- 1420 DDS_SNIPPET_OPTIMIZATION_TRANSPORT_LARGE_BUFFERS, DDS_MultiChannelQosPolicy, 1858
- 1423 DDS_SNIPPET_QOS_POLICY_BATCHING_ENABLE, check_crc
- 1423 DDS_SNIPPET_QOS_POLICY_BATCHING_ENABLE, DDS_WireProtocolQosPolicy, 2100
- 1423 DDS_SNIPPET_QOS_POLICY_DURABILITY_PERSISTENT, checkpoint_thread_settings
- 1423 DDS_SNIPPET_QOS_POLICY_DURABILITY_PERSISTENT, NDDS_Utility_NetworkCaptureParams_t, 2166
- 1422 DDS_SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT, class_id
- 1422 DDS_SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT, DDS_TransportInfo_t, 2052
- 1422 DDS_SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT_LOCAL, class_id
- 1422 DDS_SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT_LOCAL, NDDS_Transport_Property_t, 2123
- 1421 DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_ALL, cleanup_period
- 1421 DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_ALL, DDS_DatabaseQosPolicy, 1583
- 1421 DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_LAST, Clock Selection, 25
- 1421 DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_LAST, close
- 1421 DDS_SNIPPET_QOS_POLICY_PUBLISH_MODE_ASYNC, NDDS_Config_LoggerDevice, 2117
- 1421 DDS_SNIPPET_QOS_POLICY_PUBLISH_MODE_ASYNC, coherent_access
- 1420 DDS_SNIPPET_QOS_POLICY_RELIABILITY_BEST_EFFORT, DDS_PresentationQosPolicy, 1893
- 1420 DDS_SNIPPET_QOS_POLICY_RELIABILITY_BEST_EFFORT, coherent_set_info
- 1420 DDS_SNIPPET_QOS_POLICY_RELIABILITY_RELIABLE, DDS_SampleInfo, 1995
- 1420 DDS_SNIPPET_QOS_POLICY_RELIABILITY_RELIABLE, coherent_set_sequence_number
- 1427 DDS_SNIPPET_TRANSPORT_TCP_LAN_CLIENT, DDS_CoherentSetInfo_t, 1567
- 1427 DDS_SNIPPET_TRANSPORT_TCP_LAN_CLIENT, collector_initial_peers
- 1428 DDS_SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_CLIENT, DDS_MonitoringDedicatedParticipantSettings, 1837
- 1428 DDS_SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_CLIENT, comm_ports_list
- 1428 DDS_SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_SERVER, DDS_Transport_UDPv4_WAN_Property_t, 2151
- 1428 DDS_SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_SERVER, comm_ports_list_length
- 1427 DDS_SNIPPET_TRANSPORT_TCP_WAN_SYMMETRIC_CLIENT, DDS_Transport_UDPv4_WAN_Property_t, 2152
- 1427 DDS_SNIPPET_TRANSPORT_TCP_WAN_SYMMETRIC_CLIENT, Common types and functions, 1102
- 1428 DDS_SNIPPET_TRANSPORT_UDP_AVOID_IP_FRAGMENTATION, DDS_BUILTIN_TOPIC_KEY_UNKNOWN, 1111
- 1428 DDS_SNIPPET_TRANSPORT_UDP_AVOID_IP_FRAGMENTATION, DDS_BuiltinTopicKey_copy, 1107
- 1429 DDS_SNIPPET_TRANSPORT_UDP_WAN, DDS_BuiltinTopicKey_equals, 1107
- 1429 DDS_SNIPPET_TRANSPORT_UDP_WAN, DDS_BuiltinTopicKey_from_guid, 1108
- 1429 DDS_SNIPPET_TRANSPORT_UDP_WAN, DDS_BuiltinTopicKey_from_instance_handle, 1109
- 1429 DDS_SNIPPET_TRANSPORT_UDP_WAN, DDS_BuiltinTopicKey_t, 1106
- 1429 DDS_SNIPPET_TRANSPORT_UDP_WAN, DDS_BuiltinTopicKey_t_INITIALIZER, 1106
- 1429 DDS_SNIPPET_TRANSPORT_UDP_WAN, DDS_BuiltinTopicKey_to_guid, 1108
- 1429 DDS_SNIPPET_TRANSPORT_UDP_WAN, DDS_BuiltinTopicKey_to_instance_handle, 1108
- 1429 DDS_SNIPPET_TRANSPORT_UDP_WAN, DDS_LOCATOR_ADDRESS_INVALID, 1110
- 1429 DDS_SNIPPET_TRANSPORT_UDP_WAN, DDS_LOCATOR_ADDRESS_LENGTH_MAX, 1104
- builtin_discovery_plugins
- DDS_DiscoveryConfigQosPolicy, 1701
- builtin_endpoints_required_mask
- DDS_ParticipantTrustInterceptorAlgorithmInfo, 1878
- builtin_kx_endpoints_required_mask
- DDS_ParticipantTrustInterceptorAlgorithmInfo, 1878
- builtin_multicast_port_offset
- DDS_RtpsWellKnownPorts_t, 1982

- DDS_LOCATOR_INVALID, 1109
- DDS_LOCATOR_KIND_INVALID, 1109
- DDS_LOCATOR_KIND_RESERVED, 1111
- DDS_LOCATOR_KIND_SHMEM, 1110
- DDS_LOCATOR_KIND_SHMEM_510, 1110
- DDS_LOCATOR_KIND_UDPv4, 1110
- DDS_LOCATOR_KIND_UDPv4_WAN, 1110
- DDS_LOCATOR_KIND_UDPv6, 1110
- DDS_LOCATOR_KIND_UDPv6_510, 1111
- DDS_LOCATOR_PORT_INVALID, 1109
- DDS_Locator_t, 1106
- DDS_PRODUCTVERSION_UNKNOWN, 1106
- DDS_PROTOCOLVERSION, 1105
- DDS_PROTOCOLVERSION_1_0, 1104
- DDS_PROTOCOLVERSION_1_1, 1105
- DDS_PROTOCOLVERSION_1_2, 1105
- DDS_PROTOCOLVERSION_2_0, 1105
- DDS_PROTOCOLVERSION_2_1, 1105
- DDS_ProtocolVersion_t, 1106
- DDS_VENDOR_ID_LENGTH_MAX, 1105
- value, 1111
- compile
 - DDS_ContentFilter, 1572
- Compliance Configuration, 1463
 - NDDS_Config_get_rpc_compliance_mask, 1475
 - NDDS_Config_get_xtypes_compliance_mask, 1474
 - NDDS_Config_load_compliance_masks, 1475
 - NDDS_CONFIG_RPC_COMPLIANCE_MASK_DEFAULT, 1465
 - NDDS_CONFIG_RPC_COMPLIANCE_MASK_VENDOR, 1465
 - NDDS_CONFIG_RPC_USE_STANDARD_RELATED_SAMPLED_DATA, 1474
 - NDDS_Config_RpcComplianceMask, 1466
 - NDDS_Config_RpcComplianceMaskBits, 1473
 - NDDS_Config_set_rpc_compliance_mask, 1476
 - NDDS_Config_set_xtypes_compliance_mask, 1474
 - NDDS_CONFIG_XTYPES_ACCEPT_UNKNOWN_DISCRIMINATOR, 1470
 - NDDS_CONFIG_XTYPES_ACCEPT_UNKNOWN_ENUM_AS_PRIMITIVE, 1469
 - NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_DEFAULT, 1464
 - NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_VENDOR, 1464
 - NDDS_CONFIG_XTYPES_DHEADER_IN_NON_PRIMITIVE_COLLECTIONS, 1468
 - NDDS_CONFIG_XTYPES_ENCAPSULATION_OPTIONS, 1469
 - NDDS_CONFIG_XTYPES_ENUM_AS_PRIMITIVE_IN_COLLECTIONS, 1468
 - NDDS_CONFIG_XTYPES_INITIALIZE_DISCRIMINATOR_TO_DEFAULT, 1472
 - NDDS_CONFIG_XTYPES_JAVA_KEYHASH_750_BIT, 1473
 - NDDS_CONFIG_XTYPES_PARAMETER_LENGTH_WITH_PADDING, 1468
 - NDDS_CONFIG_XTYPES_PYTHON_INT8_730_BIT, 1473
 - NDDS_CONFIG_XTYPES_SELECT_DEFAULT_DISCRIMINATOR_BIT, 1470
 - NDDS_CONFIG_XTYPES_SENTINEL_IN_EMPTY_UNION_BIT, 1471
 - NDDS_Config_XTypesComplianceMask, 1465
 - NDDS_Config_XTypesComplianceMaskBits, 1467
 - compressed_sample_count
 - DDS_DataReaderCacheStatus, 1592
 - Compression Settings, 1259
 - DDS_COMPRESSION_ID_BZIP2, 1262
 - DDS_COMPRESSION_ID_LZ4, 1263
 - DDS_COMPRESSION_ID_MASK_ALL, 1260
 - DDS_COMPRESSION_ID_MASK_NONE, 1260
 - DDS_COMPRESSION_ID_MASK_PUBLICATION_DEFAULT, 1261
 - DDS_COMPRESSION_ID_MASK_SUBSCRIPTION_DEFAULT, 1261
 - DDS_COMPRESSION_ID_ZLIB, 1262
 - DDS_COMPRESSION_LEVEL_BEST_COMPRESSION, 1260
 - DDS_COMPRESSION_LEVEL_BEST_SPEED, 1260
 - DDS_COMPRESSION_LEVEL_DEFAULT, 1261
 - DDS_COMPRESSION_THRESHOLD_DEFAULT, 1261
 - DDS_CompressionId_t, 1262
 - DDS_CompressionIdMask, 1262
 - compression_ids
 - DDS_CompressionSettings_t, 1568
 - compression_settings
 - DDS_DataRepresentationQosPolicy, 1640
 - compute_crc
 - DDS_OverloadProtocolQosPolicy, 2100
 - Concurrency, 839
 - DDS_AuthenticationLevel
 - DDS_MonitoringEventDistributionSettings, 1841
 - DDS_MonitoringLoggingDistributionSettings, 1844
 - Conditions and WaitSets, 1367
 - DDS_Condition, 1370
 - DDS_Condition_dispatch, 1375
 - DDS_Condition_SetHandler, 1375
 - DDS_Condition_get_trigger_value, 1373
 - DDS_Condition_SetHandler, 1375
 - DDS_ConditionHandler_INITIALIZER, 1369
 - DDS_ConditionHandler_OnConditionTriggeredCallback, 1370
 - DDS_GuardCondition, 1370
 - DDS_GuardCondition_as_condition, 1376
 - DDS_GuardCondition_delete, 1376

- DDS_GuardCondition_new, 1376
- DDS_GuardCondition_set_trigger_value, 1377
- DDS_StatusCondition, 1371
- DDS_StatusCondition_as_condition, 1377
- DDS_StatusCondition_get_enabled_statuses, 1379
- DDS_StatusCondition_get_entity, 1380
- DDS_StatusCondition_set_enabled_statuses, 1379
- DDS_WaitSet, 1371
- DDS_WaitSet_attach_condition, 1383
- DDS_WaitSet_delete, 1381
- DDS_WaitSet_detach_condition, 1384
- DDS_WaitSet_get_conditions, 1384
- DDS_WaitSet_get_property, 1382
- DDS_WaitSet_new, 1380
- DDS_WaitSet_new_ex, 1380
- DDS_WaitSet_set_property, 1382
- DDS_WaitSet_wait, 1382
- DDS_WaitSetProperty_t_INITIALIZER, 1369
- Configuring QoS Profiles with XML, 971
- Content Filtering, 841
- content_filter_allocation
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1739
- content_filter_dropped_sample_count
 - DDS_DataReaderCacheStatus, 1588
- content_filter_hash_buckets
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1745
- content_filter_property
 - DDS_SubscriptionBuiltinTopicData, 2019
- content_filter_topic_name
 - DDS_ContentFilterProperty_t, 1579
- content_filtered_topic_allocation
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1739
- content_filtered_topic_hash_buckets
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1744
- contentfilter_property_max_length
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1751
- Conventions, 1011
- Cookie, 1385
 - DDS_Cookie_from_pointer, 1385
 - DDS_Cookie_to_pointer, 1386
- cookie
 - DDS_WriteParams_t, 2103
- corrupted_rtps_message_count
 - DDS_DomainParticipantProtocolStatus, 1726
- corrupted_rtps_message_count_change
 - DDS_DomainParticipantProtocolStatus, 1726
- count
 - DDS_QosPolicyCount, 1925
- cpu_list

- DDS_ThreadSettings_t, 2030
- cpu_rotation
 - DDS_ThreadSettings_t, 2030
- create_thread
 - DDS_ThreadFactory, 2028
- Creating Custom Content Filters, 998
- Creating New Transport Plugins, 737
- current_count
 - DDS_PublicationMatchedStatus, 1916
 - DDS_ServiceRequestAcceptedStatus, 2003
 - DDS_SubscriptionMatchedStatus, 2023
- current_count_change
 - DDS_PublicationMatchedStatus, 1916
 - DDS_ServiceRequestAcceptedStatus, 2003
 - DDS_SubscriptionMatchedStatus, 2024
- current_count_peak
 - DDS_PublicationMatchedStatus, 1916
 - DDS_SubscriptionMatchedStatus, 2023
- data
 - DDS_DynamicDataTypeProperty_t, 1781
- Data Availability, 843
- Data Samples, 700
 - DDS_CoherentSetInfo_copy, 702
 - DDS_CoherentSetInfo_equals, 701
 - DDS_SampleInfo_get_related_sample_identity, 702
 - DDS_SampleInfo_get_sample_identity, 702
- Data Writers, 471
 - DDS_DataWriter, 485
 - DDS_DataWriter_as_entity, 537
 - DDS_DataWriter_assert_liveliness, 538
 - DDS_DataWriter_flush, 555
 - DDS_DataWriter_get_datawriter_cache_status, 548
 - DDS_DataWriter_get_datawriter_protocol_status, 548
 - DDS_DataWriter_get_listener, 554
 - DDS_DataWriter_get_listenerX, 554
 - DDS_DataWriter_get_liveliness_lost_status, 545
 - DDS_DataWriter_get_matched_subscription_data, 540
 - DDS_DataWriter_get_matched_subscription_datawriter_protocol_status, 549
 - DDS_DataWriter_get_matched_subscription_datawriter_protocol_status, 549
 - DDS_DataWriter_get_matched_subscription_locators, 538
 - DDS_DataWriter_get_matched_subscription_participant_data, 542
 - DDS_DataWriter_get_matched_subscriptions, 539
 - DDS_DataWriter_get_offered_deadline_missed_status, 546
 - DDS_DataWriter_get_offered_incompatible_qos_status, 546

- DDS_DataWriter_get_publication_matched_status, 546
- DDS_DataWriter_get_publisher, 543
- DDS_DataWriter_get_qos, 552
- DDS_DataWriter_get_reliable_reader_activity_changed_status, 547
- DDS_DataWriter_get_reliable_writer_cache_changed_status, 547
- DDS_DataWriter_get_service_request_accepted_status, 550
- DDS_DataWriter_get_topic, 542
- DDS_DataWriter_is_matched_subscription_active, 540
- DDS_DataWriter_is_sample_app_acknowledged, 544
- DDS_DataWriter_set_listener, 553
- DDS_DataWriter_set_property, 552
- DDS_DataWriter_set_qos, 550
- DDS_DataWriter_set_qos_with_profile, 551
- DDS_DataWriter_take_discovery_snapshot, 555
- DDS_DataWriter_wait_for_acknowledgments, 543
- DDS_DataWriter_wait_for_asynchronous_publishing, 544
- DDS_DataWriterCacheStatus_copy, 529
- DDS_DataWriterCacheStatus_equals, 530
- DDS_DataWriterCacheStatus_finalize, 529
- DDS_DataWriterCacheStatus_initialize, 528
- DDS_DataWriterCacheStatus_INITIALIZER, 483
- DDS_DataWriterListener_INITIALIZER, 484
- DDS_DataWriterListener_InstanceReplacedCallback, 489
- DDS_DataWriterListener_LivelinessLostCallback, 486
- DDS_DataWriterListener_OfferedDeadlineMissedCallback, 486
- DDS_DataWriterListener_OfferedIncompatibleQosCallback, 487
- DDS_DataWriterListener_OnApplicationAcknowledgmentCallback, 489
- DDS_DataWriterListener_PublicationMatchedCallback, 487
- DDS_DataWriterListener_ReliableReaderActivityChangedCallback, 488
- DDS_DataWriterListener_ReliableWriterCacheChangedCallback, 487
- DDS_DataWriterListener_SampleRemovedCallback, 488
- DDS_DataWriterListener_ServiceRequestAcceptedCallback, 489
- DDS_DataWriterProtocolStatus_copy, 531
- DDS_DataWriterProtocolStatus_equals, 532
- DDS_DataWriterProtocolStatus_finalize, 532
- DDS_DataWriterProtocolStatus_initialize, 531
- DDS_DataWriterProtocolStatus_INITIALIZER, 483
- DDS_DataWriterQos_copy, 537
- DDS_DataWriterQos_equals, 533
- DDS_DataWriterQos_finalize, 536
- DDS_DataWriterQos_initialize, 535
- DDS_DataWriterQos_INITIALIZER, 484
- DDS_DataWriterQos_print, 533
- DDS_DataWriterQos_to_string, 534
- DDS_DataWriterQos_to_string_w_params, 534
- DDS_LivelinessLostStatus_copy, 513
- DDS_LivelinessLostStatus_equals, 515
- DDS_LivelinessLostStatus_finalize, 514
- DDS_LivelinessLostStatus_initialize, 513
- DDS_LivelinessLostStatus_INITIALIZER, 480
- DDS_OfferedDeadlineMissedStatus_copy, 511
- DDS_OfferedDeadlineMissedStatus_equals, 512
- DDS_OfferedDeadlineMissedStatus_finalize, 512
- DDS_OfferedDeadlineMissedStatus_initialize, 510
- DDS_OfferedDeadlineMissedStatus_INITIALIZER, 479
- DDS_OfferedIncompatibleQosStatus_copy, 516
- DDS_OfferedIncompatibleQosStatus_equals, 517
- DDS_OfferedIncompatibleQosStatus_finalize, 516
- DDS_OfferedIncompatibleQosStatus_initialize, 515
- DDS_OfferedIncompatibleQosStatus_INITIALIZER, 480
- DDS_PublicationMatchedStatus_copy, 518
- DDS_PublicationMatchedStatus_equals, 519
- DDS_PublicationMatchedStatus_finalize, 519
- DDS_PublicationMatchedStatus_initialize, 518
- DDS_PublicationMatchedStatus_INITIALIZER, 481
- DDS_ReliableReaderActivityChangedStatus_copy, 526
- DDS_ReliableReaderActivityChangedStatus_equals, 528
- DDS_ReliableReaderActivityChangedStatus_finalize, 526
- DDS_ReliableReaderActivityChangedStatus_initialize, 525
- DDS_ReliableReaderActivityChangedStatus_INITIALIZER, 482
- DDS_ReliableWriterCacheChangedStatus_copy, 523
- DDS_ReliableWriterCacheChangedStatus_equals, 525
- DDS_ReliableWriterCacheChangedStatus_finalize, 524
- DDS_ReliableWriterCacheChangedStatus_initialize, 523
- DDS_ReliableWriterCacheChangedStatus_INITIALIZER, 482
- DDS_ReliableWriterCacheEventCount_equals, 522
- DDS_ServiceRequestAcceptedStatus_copy, 521
- DDS_ServiceRequestAcceptedStatus_equals, 522
- DDS_ServiceRequestAcceptedStatus_finalize, 521

- DDS_ServiceRequestAcceptedStatus_initialize, 520
- DDS_ServiceRequestAcceptedStatus_INITIALIZER, 481
- FooDataWriter_as_datawriter, 490
- FooDataWriter_create_data, 506
- FooDataWriter_create_data_w_params, 507
- FooDataWriter_delete_data, 507
- FooDataWriter_delete_data_w_params, 508
- FooDataWriter_discard_loan, 510
- FooDataWriter_dispose, 502
- FooDataWriter_dispose_w_params, 504
- FooDataWriter_dispose_w_timestamp, 503
- FooDataWriter_get_key_value, 505
- FooDataWriter_get_loan, 508
- FooDataWriter_lookup_instance, 506
- FooDataWriter_narrow, 490
- FooDataWriter_register_instance, 491
- FooDataWriter_register_instance_w_params, 493
- FooDataWriter_register_instance_w_timestamp, 492
- FooDataWriter_unregister_instance, 493
- FooDataWriter_unregister_instance_w_params, 496
- FooDataWriter_unregister_instance_w_timestamp, 495
- FooDataWriter_write, 496
- FooDataWriter_write_w_params, 501
- FooDataWriter_write_w_timestamp, 500
- DATA_READER_PROTOCOL, 1253
 - DDS_DATAREADERPROTOCOL_QOS_POLICY_NAME, 1253
- DATA_READER_RESOURCE_LIMITS, 1253
 - DDS_ANY_INSTANCE_REMOVAL, 1255
 - DDS_AUTO_MAX_TOTAL_INSTANCES, 1256
 - DDS_DataReaderInstanceRemovalKind, 1254
 - DDS_DATAREADERRESOURCELIMITS_QOS_POLICY_NAME, 1256
 - DDS_EMPTY_INSTANCE_REMOVAL, 1255
 - DDS_FULLY_PROCESSED_INSTANCE_REMOVAL, 1255
 - DDS_NO_INSTANCE_REMOVAL, 1255
- DATA_REPRESENTATION, 1256
 - DDS_AUTO_DATA_REPRESENTATION, 1258
 - DDS_DATA_REPRESENTATION_QOS_POLICY_NAME, 1258
 - DDS_DataRepresentationId_t, 1257
 - DDS_XCDR2_DATA_REPRESENTATION, 1258
 - DDS_XCDR_DATA_REPRESENTATION, 1257
 - DDS_XML_DATA_REPRESENTATION, 1257
- DATA_TAG, 1263
 - DDS_DATATAG_QOS_POLICY_NAME, 1268
 - DDS_DataTagQosPolicy, 1264
 - DDS_DataTagQosPolicyHelper_add_tag, 1266
 - DDS_DataTagQosPolicyHelper_assert_tag, 1265
 - DDS_DataTagQosPolicyHelper_get_number_of_tags, 1267
 - DDS_DataTagQosPolicyHelper_lookup_tag, 1265
 - DDS_DataTagQosPolicyHelper_remove_tag, 1266
- data_tags
 - DDS_DataReaderQos, 1615
 - DDS_DataWriterQos, 1670
 - DDS_PublicationBuiltinTopicData, 1910
 - DDS_SubscriptionBuiltinTopicData, 2017
- DATA_WRITER_PROTOCOL, 1268
 - DDS_DATAWRITERPROTOCOL_QOS_POLICY_NAME, 1268
- DATA_WRITER_RESOURCE_LIMITS, 1269
 - DDS_ALIVE_INSTANCE_REPLACEMENT, 1271
 - DDS_ALIVE_OR_DISPOSED_INSTANCE_REPLACEMENT, 1271
 - DDS_ALIVE_THEN_DISPOSED_INSTANCE_REPLACEMENT, 1271
 - DDS_DATAWRITERRESOURCELIMITS_QOS_POLICY_NAME, 1271
 - DDS_DataWriterResourceLimitsInstanceReplacementKind, 1269
 - DDS_DISPOSED_INSTANCE_REPLACEMENT, 1271
 - DDS_DISPOSED_THEN_ALIVE_INSTANCE_REPLACEMENT, 1271
 - DDS_UNREGISTERED_INSTANCE_REPLACEMENT, 1270
- DATA_WRITER_TRANSFER_MODE, 1272
 - DDS_DATAWRITERTRANSFERMODE_QOS_POLICY_NAME, 1272
- DATABASE, 1252
 - DDS_DATABASE_QOS_POLICY_NAME, 1252
- database
 - DDS_DomainParticipantQos, 1730
- DataReader Use Cases, 985
- datareader_qos
 - RTI_Connext_ReplierParams, 2171
 - RTI_Connext_RequesterParams, 2176
 - RTI_Connext_SimpleReplierParams, 2180
- DataReaders, 601
 - DDS_DataReader, 615
 - DDS_DataReader_acknowledge_all, 677
 - DDS_DataReader_acknowledge_all_w_response, 676
 - DDS_DataReader_acknowledge_sample, 677
 - DDS_DataReader_acknowledge_sample_w_response, 675
 - DDS_DataReader_as_entity, 671
 - DDS_DataReader_create_querycondition, 672
 - DDS_DataReader_create_querycondition_w_params, 673
 - DDS_DataReader_create_readcondition, 671
 - DDS_DataReader_create_readcondition_w_params, 672
 - DDS_DataReader_create_topic_query, 691

- DDS_DataReader_delete_contained_entities, 674
- DDS_DataReader_delete_readcondition, 673
- DDS_DataReader_delete_topic_query, 692
- DDS_DataReader_get_datareader_cache_status, 685
- DDS_DataReader_get_datareader_protocol_status, 686
- DDS_DataReader_get_listener, 690
- DDS_DataReader_get_listenerX, 691
- DDS_DataReader_get_liveliness_changed_status, 682
- DDS_DataReader_get_matched_publication_data, 679
- DDS_DataReader_get_matched_publication_datareader_protocol_status, 686
- DDS_DataReader_get_matched_publication_participant_data, 680
- DDS_DataReader_get_matched_publications, 678
- DDS_DataReader_get_qos, 688
- DDS_DataReader_get_requested_deadline_missed_status, 682
- DDS_DataReader_get_requested_incompatible_qos_status, 683
- DDS_DataReader_get_sample_lost_status, 685
- DDS_DataReader_get_sample_rejected_status, 682
- DDS_DataReader_get_subscriber, 681
- DDS_DataReader_get_subscription_matched_status, 683
- DDS_DataReader_get_topicdescription, 681
- DDS_DataReader_is_matched_publication_alive, 679
- DDS_DataReader_lookup_topic_query, 692
- DDS_DataReader_set_listener, 690
- DDS_DataReader_set_property, 689
- DDS_DataReader_set_qos, 687
- DDS_DataReader_set_qos_with_profile, 687
- DDS_DataReader_take_discovery_snapshot, 693
- DDS_DataReader_wait_for_historical_data, 675
- DDS_DataReaderCacheStatus_copy, 661
- DDS_DataReaderCacheStatus_equals, 663
- DDS_DataReaderCacheStatus_finalize, 662
- DDS_DataReaderCacheStatus_initialize, 661
- DDS_DataReaderCacheStatus_INITIALIZER, 613
- DDS_DataReaderListener_DataAvailableCallback, 617
- DDS_DataReaderListener_INITIALIZER, 615
- DDS_DataReaderListener_LivelinessChangedCallback, 617
- DDS_DataReaderListener_RequestedDeadlineMissedCallback, 616
- DDS_DataReaderListener_RequestedIncompatibleQosCallback, 617
- DDS_DataReaderListener_SampleLostCallback, 618
- DDS_DataReaderListener_SampleRejectedCallback, 617
- DDS_DataReaderListener_SubscriptionMatchedCallback, 617
- DDS_DataReaderProtocolStatus_copy, 664
- DDS_DataReaderProtocolStatus_equals, 666
- DDS_DataReaderProtocolStatus_finalize, 664
- DDS_DataReaderProtocolStatus_initialize, 663
- DDS_DataReaderProtocolStatus_INITIALIZER, 614
- DDS_DataReaderQos_copy, 669
- DDS_DataReaderQos_equals, 666
- DDS_DataReaderQos_finalize, 671
- DDS_DataReaderQos_initialize, 669
- DDS_DataReaderQos_INITIALIZER, 614
- DDS_DataReaderQos_print, 667
- DDS_DataReaderQos_to_string, 667
- DDS_DataReaderQos_to_string_w_params, 668
- DDS_LivelinessChangedStatus_copy, 652
- DDS_LivelinessChangedStatus_equals, 653
- DDS_LivelinessChangedStatus_finalize, 652
- DDS_LivelinessChangedStatus_initialize, 651
- DDS_LivelinessChangedStatus_INITIALIZER, 611
- DDS_LOST_BY_AVAILABILITY_WAITING_TIME, 620
- DDS_LOST_BY_DECODE_FAILURE, 621
- DDS_LOST_BY_DESERIALIZATION_FAILURE, 621
- DDS_LOST_BY_INCOMPLETE_COHERENT_SET, 619
- DDS_LOST_BY_INSTANCES_LIMIT, 618
- DDS_LOST_BY_LARGE_COHERENT_SET, 619
- DDS_LOST_BY_OUT_OF_MEMORY, 621
- DDS_LOST_BY_REMOTE_WRITER_SAMPLES_PER_VIRTUAL_QUEUE_LIMIT, 621
- DDS_LOST_BY_REMOTE_WRITERS_PER_INSTANCE_LIMIT, 619
- DDS_LOST_BY_REMOTE_WRITERS_PER_SAMPLE_LIMIT, 620
- DDS_LOST_BY_SAMPLES_LIMIT, 622
- DDS_LOST_BY_SAMPLES_PER_INSTANCE_LIMIT, 621
- DDS_LOST_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT, 620
- DDS_LOST_BY_UNKNOWN_INSTANCE, 621
- DDS_LOST_BY_VIRTUAL_WRITERS_LIMIT, 620
- DDS_LOST_BY_WRITER, 618
- DDS_NOT_LOST, 618
- DDS_NOT_REJECTED, 622
- DDS_REJECTED_BY_DECODE_FAILURE, 624
- DDS_REJECTED_BY_INSTANCES_LIMIT, 622
- DDS_REJECTED_BY_REMOTE_WRITER_SAMPLES_PER_VIRTUAL_QUEUE_LIMIT, 623
- DDS_REJECTED_BY_SAMPLES_LIMIT, 622
- DDS_REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT, 623
- DDS_REJECTED_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT, 623

- 623
- DDS_RequestedDeadlineMissedStatus_copy, 650
- DDS_RequestedDeadlineMissedStatus_equals, 651
- DDS_RequestedDeadlineMissedStatus_finalize, 651
- DDS_RequestedDeadlineMissedStatus_initialize, 650
- DDS_RequestedDeadlineMissedStatus_INITIALIZER, 610
- DDS_RequestedIncompatibleQosStatus_copy, 654
- DDS_RequestedIncompatibleQosStatus_equals, 655
- DDS_RequestedIncompatibleQosStatus_finalize, 654
- DDS_RequestedIncompatibleQosStatus_initialize, 653
- DDS_RequestedIncompatibleQosStatus_INITIALIZER, 611
- DDS_SampleLostStatus_copy, 656
- DDS_SampleLostStatus_equals, 657
- DDS_SampleLostStatus_finalize, 656
- DDS_SampleLostStatus_initialize, 655
- DDS_SampleLostStatus_INITIALIZER, 612
- DDS_SampleLostStatusKind, 618
- DDS_SampleRejectedStatus_copy, 658
- DDS_SampleRejectedStatus_equals, 659
- DDS_SampleRejectedStatus_finalize, 658
- DDS_SampleRejectedStatus_initialize, 657
- DDS_SampleRejectedStatus_INITIALIZER, 612
- DDS_SampleRejectedStatusKind, 622
- DDS_SubscriptionMatchedStatus_copy, 660
- DDS_SubscriptionMatchedStatus_equals, 660
- DDS_SubscriptionMatchedStatus_finalize, 660
- DDS_SubscriptionMatchedStatus_initialize, 659
- DDS_SubscriptionMatchedStatus_INITIALIZER, 613
- FooDataReader_as_datareader, 624
- FooDataReader_get_key_value, 647
- FooDataReader_is_data_consistent, 649
- FooDataReader_lookup_instance, 648
- FooDataReader_narrow, 624
- FooDataReader_read, 625
- FooDataReader_read_instance, 635
- FooDataReader_read_instance_w_condition, 638
- FooDataReader_read_next_instance, 640
- FooDataReader_read_next_instance_w_condition, 643
- FooDataReader_read_next_sample, 633
- FooDataReader_read_w_condition, 630
- FooDataReader_return_loan, 646
- FooDataReader_take, 626
- FooDataReader_take_instance, 636
- FooDataReader_take_instance_w_condition, 639
- FooDataReader_take_next_instance, 642
- FooDataReader_take_next_instance_w_condition, 645
- FooDataReader_take_next_sample, 634
- FooDataReader_take_w_condition, 632
- DataWriter Use Cases, 982
- datawriter_qos
 - RTI_Connext_ReplierParams, 2171
 - RTI_Connext_RequesterParams, 2175
 - RTI_Connext_SimpleReplierParams, 2180
- datawriter_qos_profile_name
 - DDS_MonitoringEventDistributionSettings, 1842
 - DDS_MonitoringLoggingDistributionSettings, 1845
 - DDS_MonitoringPeriodicDistributionSettings, 1852
- DDS Extension QoS, 795
- DDS Specification QoS, 790
- DDS-Specific Primitive Types, 1199
 - DDS_Boolean, 1204
 - DDS_BOOLEAN_FALSE, 1201
 - DDS_BOOLEAN_TRUE, 1201
 - DDS_Char, 1201
 - DDS_Double, 1203
 - DDS_Enum, 1204
 - DDS_Float, 1203
 - DDS_Int8, 1202
 - DDS_Long, 1203
 - DDS_LongDouble, 1204
 - DDS_LongLong, 1203
 - DDS-Octet, 1202
 - DDS_Short, 1202
 - DDS_UInt8, 1202
 - DDS_UnsignedLong, 1203
 - DDS_UnsignedLongLong, 1203
 - DDS_UnsignedShort, 1202
 - DDS_Wchar, 1202
- DDS_AcknowledgmentInfo, 1535
 - response_data, 1536
 - sample_identity, 1536
 - subscription_handle, 1535
 - valid_response_data, 1536
- DDS_AckResponseData_t, 1536
 - value, 1537
- DDS_ADMIN_CONSOLE_SERVICE_QOS
 - SERVICE, 1327
- DDS_ALIVE_INSTANCE_REPLACEMENT
 - DATA_WRITER_RESOURCE_LIMITS, 1271
- DDS_ALIVE_INSTANCE_STATE
 - Instance States, 715
- DDS_ALIVE_OR_DISPOSED_INSTANCE_REPLACEMENT
 - DATA_WRITER_RESOURCE_LIMITS, 1271
- DDS_ALIVE_THEN_DISPOSED_INSTANCE_REPLACEMENT
 - DATA_WRITER_RESOURCE_LIMITS, 1271
- DDS_AllocationSettings_t, 1537
 - incremental_count, 1538
 - initial_count, 1538
 - max_count, 1538
- DDS_ALLOW_TYPE_COERCION

- TYPE_CONSISTENCY_ENFORCEMENT, 1341
- DDS_AnnotationParameterValue, 1538
- DDS_ANY_INSTANCE_REMOVAL
 - DATA_READER_RESOURCE_LIMITS, 1255
- DDS_ANY_INSTANCE_STATE
 - Instance States, 715
- DDS_ANY_SAMPLE_STATE
 - Sample States, 711
- DDS_ANY_VIEW_STATE
 - View States, 713
- DDS_APPLICATION_AUTO_ACKNOWLEDGMENT_MODE
 - RELIABILITY, 1323
- DDS_APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE
 - RELIABILITY, 1324
- DDS_ASYNC_WAITSET_COMPLETION_TOKEN_IGNORE
 - AsyncWaitSet, 1090
- DDS_ASYNC_WAITSET_COMPLETION_TOKEN_USE_IMPLICIT_AND_WAITSET
 - AsyncWaitSet, 1090
- DDS_ASYNC_WAITSET_PROPERTY_DEFAULT
 - AsyncWaitSet, 1090
- DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS
 - PUBLISH_MODE, 1319
- DDS_ASYNCHRONOUSPUBLISHER_QOS_POLICY_ID
 - All QoS Policies, 1248
- DDS_ASYNCHRONOUSPUBLISHER_QOS_POLICY_NAME
 - ASYNCHRONOUS_PUBLISHER, 1250
- DDS_AsyncronousPublisherQosPolicy, 1539
 - asynchronous_batch_thread, 1541
 - disable_asynchronous_batch, 1541
 - disable_asynchronous_write, 1540
 - disable_topic_query_publication, 1542
 - thread, 1540
 - topic_query_publication_thread, 1542
- DDS_AsyncWaitSet
 - AsyncWaitSet, 1070
- DDS_AsyncWaitSet_attach_condition
 - AsyncWaitSet, 1078
- DDS_AsyncWaitSet_attach_condition_with_completion_token
 - AsyncWaitSet, 1079
- DDS_AsyncWaitSet_create_completion_token
 - AsyncWaitSet, 1085
- DDS_AsyncWaitSet_delete
 - AsyncWaitSet, 1088
- DDS_AsyncWaitSet_delete_completion_token
 - AsyncWaitSet, 1085
- DDS_AsyncWaitSet_detach_condition
 - AsyncWaitSet, 1077
- DDS_AsyncWaitSet_detach_condition_with_completion_token
 - AsyncWaitSet, 1077
- DDS_AsyncWaitSet_get_conditions
 - AsyncWaitSet, 1080
- DDS_AsyncWaitSet_get_property
 - AsyncWaitSet, 1076
- DDS_AsyncWaitSet_is_started
 - AsyncWaitSet, 1083
- DDS_AsyncWaitSet_new
 - AsyncWaitSet, 1086
- DDS_AsyncWaitSet_new_with_listener
 - AsyncWaitSet, 1086
- DDS_AsyncWaitSet_new_with_thread_factory
 - AsyncWaitSet, 1088
- DDS_AsyncWaitSet_start
 - AsyncWaitSet, 1082
- DDS_AsyncWaitSet_start_with_completion_token
 - AsyncWaitSet, 1082
- DDS_AsyncWaitSet_stop
 - AsyncWaitSet, 1081
- DDS_AsyncWaitSet_stop_with_completion_token
 - AsyncWaitSet, 1081
- DDS_AsyncWaitSet_unlock_condition
 - AsyncWaitSet, 1080
- DDS_AsyncWaitSetCompletionToken
 - AsyncWaitSet, 1073
- DDS_AsyncWaitSetCompletionToken_wait
 - AsyncWaitSet, 1076
- DDS_AsyncWaitSetListener, 1543
 - listener_data, 1544
 - on_thread_deleted, 1544
 - on_thread_spawned, 1544
 - on_wait_timeout, 1544
- DDS_AsyncWaitSetListener_INITIALIZER
 - AsyncWaitSet, 1070
- DDS_AsyncWaitSetListener_OnThreadDeletedCallback
 - AsyncWaitSet, 1074
- DDS_AsyncWaitSetListener_OnThreadSpawnedCallback
 - AsyncWaitSet, 1074
- DDS_AsyncWaitSetListener_OnWaitTimeoutCallback
 - AsyncWaitSet, 1075
- DDS_AsyncWaitSetProperty_t, 1544
 - level, 1547
 - thread_name_prefix, 1546
 - thread_pool_size, 1545
 - thread_settings, 1546
 - wait_timeout, 1546
 - waitset_property, 1545
- DDS_AUTO_CDR_PADDING
 - TYPESUPPORT, 1342
- DDS_AUTO_COUNT
 - DOMAIN_PARTICIPANT_RESOURCE_LIMITS, 1284
- DDS_AUTO_DATA_REPRESENTATION
 - DATA_REPRESENTATION, 1258
- DDS_AUTO_MAX_TOTAL_INSTANCES
 - DATA_READER_RESOURCE_LIMITS, 1256
- DDS_AUTO_SAMPLE_IDENTITY
 - WriteParams, 1391
- DDS_AUTO_SEQUENCE_NUMBER
 - Sequence Number Support, 1218

- DDS_AUTO_TYPE_COERCION
 - TYPE_CONSISTENCY_ENFORCEMENT, 1341
- DDS_AUTO_WRITER_DEPTH
 - DURABILITY, 1288
- DDS_AUTOMATIC_LIVELINESS_QOS
 - LIVELINESS, 1297
- DDS_AUTOMATIC_TRANSPORT_MULTICAST_QOS
 - TRANSPORT_MULTICAST, 1336
- DDS_AVAILABILITY_QOS_POLICY_ID
 - All QoS Policies, 1248
- DDS_AVAILABILITY_QOS_POLICY_NAME
 - AVAILABILITY, 1251
- DDS_AvailabilityQosPolicy, 1547
 - enable_required_subscriptions, 1550
 - max_data_availability_waiting_time, 1550
 - max_endpoint_availability_waiting_time, 1550
 - required_matched_endpoint_groups, 1551
- DDS_BAD_MEMBER_ID_USER_EXCEPTION_CODE
 - Exception Codes, 1220
- DDS_BAD_MEMBER_NAME_USER_EXCEPTION_CODE
 - Exception Codes, 1220
- DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE
 - Exception Codes, 1220
- DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE
 - Exception Codes, 1220
- DDS_BADKIND_USER_EXCEPTION_CODE
 - Exception Codes, 1220
- DDS_BATCH_QOS_POLICY_ID
 - All QoS Policies, 1248
- DDS_BATCH_QOS_POLICY_NAME
 - BATCH, 1252
- DDS_BatchQosPolicy, 1552
 - enable, 1552
 - max_data_bytes, 1553
 - max_flush_delay, 1554
 - max_samples, 1553
 - source_timestamp_resolution, 1554
 - thread_safe_write, 1555
- DDS_BEST_EFFORT_RELIABILITY_QOS
 - RELIABILITY, 1323
- DDS_Boolean
 - DDS-Specific Primitive Types, 1204
- DDS_BOOLEAN_FALSE
 - DDS-Specific Primitive Types, 1201
- DDS_BOOLEAN_TRUE
 - DDS-Specific Primitive Types, 1201
- DDS_BooleanSeq, 1556
- DDS_BOUNDS_USER_EXCEPTION_CODE
 - Exception Codes, 1220
- dds_builtin_endpoints
 - DDS_ParticipantBuiltinTopicData, 1872
- DDS_BUILTIN_QOS_LIB
 - Builtin Qos Profiles, 1400
- DDS_BUILTIN_QOS_LIB_EXP
 - Builtin Qos Profiles, 1404
- DDS_BUILTIN_QOS_SNIPPET_LIB
 - Builtin Qos Profiles, 1415
- DDS_BUILTIN_TOPIC_KEY_UNKNOWN
 - Common types and functions, 1111
- DDS_BuiltinTopicKey_copy
 - Common types and functions, 1107
- DDS_BuiltinTopicKey_equals
 - Common types and functions, 1107
- DDS_BuiltinTopicKey_from_guid
 - Common types and functions, 1108
- DDS_BuiltinTopicKey_from_instance_handle
 - Common types and functions, 1109
- DDS_BuiltinTopicKey_t, 1556
 - Common types and functions, 1106
- DDS_BuiltinTopicKey_t_INITIALIZER
 - Common types and functions, 1106
- DDS_BuiltinTopicKey_to_guid
 - Common types and functions, 1108
- DDS_BuiltinTopicKey_to_instance_handle
 - Common types and functions, 1108
- DDS_BuiltinTopicReaderResourceLimits_t, 1557
 - disable_fragmentation_support, 1560
 - dynamically_allocate_fragmented_samples, 1563
 - initial_fragmented_samples, 1561
 - initial_infos, 1559
 - initial_outstanding_reads, 1559
 - initial_samples, 1558
 - max_fragmented_samples, 1561
 - max_fragmented_samples_per_remote_writer, 1562
 - max_fragments_per_sample, 1562
 - max_infos, 1559
 - max_outstanding_reads, 1560
 - max_samples, 1558
 - max_samples_per_read, 1560
- DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS
 - DESTINATION_ORDER, 1274
- DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS
 - DESTINATION_ORDER, 1274
- DDS_CdrPaddingKind
 - TYPESUPPORT, 1342
- DDS_ChannelSettings_t, 1564
 - filter_expression, 1564
 - multicast_settings, 1564
 - priority, 1565
- DDS_ChannelSettingsSeq, 1566
- DDS_Char
 - DDS-Specific Primitive Types, 1201
- DDS_CharSeq, 1566
- DDS_CoherentSetInfo_copy
 - Data Samples, 702
- DDS_CoherentSetInfo_equals
 - Data Samples, 701
- DDS_CoherentSetInfo_t, 1566

- coherent_set_sequence_number, 1567
- group_coherent_set_sequence_number, 1567
- group_guid, 1567
- incomplete_coherent_set, 1568
- DDS_COMPRESSION_ID_BZIP2
 - Compression Settings, 1262
- DDS_COMPRESSION_ID_LZ4
 - Compression Settings, 1263
- DDS_COMPRESSION_ID_MASK_ALL
 - Compression Settings, 1260
- DDS_COMPRESSION_ID_MASK_NONE
 - Compression Settings, 1260
- DDS_COMPRESSION_ID_MASK_PUBLICATION_DEFAULT
 - Compression Settings, 1261
- DDS_COMPRESSION_ID_MASK_SUBSCRIPTION_DEFAULT
 - Compression Settings, 1261
- DDS_COMPRESSION_ID_ZLIB
 - Compression Settings, 1262
- DDS_COMPRESSION_LEVEL_BEST_COMPRESSION
 - Compression Settings, 1260
- DDS_COMPRESSION_LEVEL_BEST_SPEED
 - Compression Settings, 1260
- DDS_COMPRESSION_LEVEL_DEFAULT
 - Compression Settings, 1261
- DDS_COMPRESSION_THRESHOLD_DEFAULT
 - Compression Settings, 1261
- DDS_CompressionId_t
 - Compression Settings, 1262
- DDS_CompressionIdMask
 - Compression Settings, 1262
- DDS_CompressionSettings_t, 1568
 - compression_ids, 1568
 - writer_compression_level, 1569
 - writer_compression_threshold, 1569
- DDS_Condition
 - Conditions and WaitSets, 1370
- DDS_Condition_dispatch
 - Conditions and WaitSets, 1375
- DDS_Condition_get_handler
 - Conditions and WaitSets, 1375
- DDS_Condition_get_trigger_value
 - Conditions and WaitSets, 1373
- DDS_Condition_set_handler
 - Conditions and WaitSets, 1375
- DDS_ConditionHandler, 1570
 - handler_data, 1571
 - on_condition_triggered, 1571
- DDS_ConditionHandler_INITIALIZER
 - Conditions and WaitSets, 1369
- DDS_ConditionHandler_OnConditionTriggeredCallback
 - Conditions and WaitSets, 1370
- DDS_ConditionSeq, 1571
- DDS_ContentFilter, 1571
 - compile, 1572
 - evaluate, 1574
 - filter_data, 1578
 - finalize, 1576
 - writer_attach, 1577
 - writer_compile, 1573
 - writer_detach, 1578
 - writer_evaluate, 1576
 - writer_finalize, 1577
 - writer_return_loan, 1578
- DDS_ContentFilter_INITIALIZER
 - Topics, 174
- DDS_ContentFilterCompileFunction
 - Topics, 177
- DDS_ContentFilteredTopic
 - Topics, 177
- DDS_ContentFilteredTopic_append_to_expression_parameter
 - Topics, 203
- DDS_ContentFilteredTopic_as_topicdescription
 - Topics, 201
- DDS_ContentFilteredTopic_get_expression_parameters
 - Topics, 202
- DDS_ContentFilteredTopic_get_filter_expression
 - Topics, 201
- DDS_ContentFilteredTopic_get_related_topic
 - Topics, 206
- DDS_ContentFilteredTopic_narrow
 - Topics, 201
- DDS_ContentFilteredTopic_remove_from_expression_parameter
 - Topics, 204
- DDS_ContentFilteredTopic_set_expression
 - Topics, 203
- DDS_ContentFilteredTopic_set_expression_parameters
 - Topics, 202
- DDS_ContentFilterEvaluateFunction
 - Topics, 178
- DDS_ContentFilterFinalizeFunction
 - Topics, 179
- DDS_ContentFilterProperty_t, 1579
 - content_filter_topic_name, 1579
 - expression_parameters, 1580
 - filter_class_name, 1579
 - filter_expression, 1580
 - related_topic_name, 1579
- DDS_ContentFilterWriterAttachFunction
 - Topics, 180
- DDS_ContentFilterWriterCompileFunction
 - Topics, 182
- DDS_ContentFilterWriterDetachFunction
 - Topics, 181
- DDS_ContentFilterWriterEvaluateFunction
 - Topics, 183
- DDS_ContentFilterWriterFinalizeFunction
 - Topics, 180
- DDS_ContentFilterWriterReturnLoanFunction

- Topics, 183
- DDS_Cookie_from_pointer
 - Cookie, 1385
- DDS_Cookie_t, 1580
 - value, 1580
- DDS_Cookie_to_pointer
 - Cookie, 1386
- DDS_CookieSeq, 1581
- DDS_DATA_AVAILABLE_STATUS
 - Status Kinds, 1230
- DDS_DATA_ON_READERS_STATUS
 - Status Kinds, 1230
- DDS_DATA_READER_CACHE_STATUS
 - Status Kinds, 1235
- DDS_DATA_READER_PROTOCOL_STATUS
 - Status Kinds, 1235
- DDS_DATA_REPRESENTATION_QOS_POLICY_ID
 - All QoS Policies, 1247
- DDS_DATA_REPRESENTATION_QOS_POLICY_NAME
 - DATA_REPRESENTATION, 1258
- DDS_DATA_WRITER_APPLICATION_ACKNOWLEDGMENT_STATUS
 - Status Kinds, 1233
- DDS_DATA_WRITER_CACHE_STATUS
 - Status Kinds, 1234
- DDS_DATA_WRITER_EMPTY_VIRTUAL_WRITER_STATUS
 - Status Kinds, 1232
- DDS_DATA_WRITER_INSTANCE_REPLACED_STATUS
 - Status Kinds, 1233
- DDS_DATA_WRITER_PROTOCOL_STATUS
 - Status Kinds, 1234
- DDS_DATA_WRITER_SAMPLE_REMOVED_STATUS
 - Status Kinds, 1235
- DDS_DATABASE_INTEGRATION_SERVICE_QOS
 - SERVICE, 1327
- DDS_DATABASE_QOS_POLICY_ID
 - All QoS Policies, 1248
- DDS_DATABASE_QOS_POLICY_NAME
 - DATABASE, 1252
- DDS_DatabaseQosPolicy, 1581
 - cleanup_period, 1583
 - initial_records, 1584
 - initial_weak_references, 1585
 - max_skiplist_level, 1584
 - max_weak_references, 1584
 - shutdown_cleanup_period, 1583
 - shutdown_timeout, 1583
 - thread, 1582
- DDS_DataReader
 - DataReaders, 615
- DDS_DataReader_acknowledge_all
 - DataReaders, 677
- DDS_DataReader_acknowledge_all_w_response
 - DataReaders, 676
- DDS_DataReader_acknowledge_sample
 - DataReaders, 677
- DDS_DataReader_acknowledge_sample_w_response
 - DataReaders, 675
- DDS_DataReader_as_entity
 - DataReaders, 671
- DDS_DATAREADER_C
 - User Data Type Support, 213
- DDS_DataReader_create_querycondition
 - DataReaders, 672
- DDS_DataReader_create_querycondition_w_params
 - DataReaders, 673
- DDS_DataReader_create_readcondition
 - DataReaders, 671
- DDS_DataReader_create_readcondition_w_params
 - DataReaders, 672
- DDS_DataReader_create_topic_query
 - DataReaders, 691
- DDS_DataReader_delete_contained_entities
 - DataReaders, 674
- DDS_DataReader_delete_readcondition
 - DataReaders, 673
- DDS_DataReader_delete_topic_query
 - DataReaders, 692
- DDS_DataReader_get_datareader_cache_status
 - DataReaders, 685
- DDS_DataReader_get_datareader_protocol_status
 - DataReaders, 686
- DDS_DataReader_get_listener
 - DataReaders, 690
- DDS_DataReader_get_listenerX
 - DataReaders, 691
- DDS_DataReader_get_liveliness_changed_status
 - DataReaders, 682
- DDS_DataReader_get_matched_publication_data
 - DataReaders, 679
- DDS_DataReader_get_matched_publication_datareader_protocol_status
 - DataReaders, 686
- DDS_DataReader_get_matched_publication_participant_data
 - DataReaders, 680
- DDS_DataReader_get_matched_publications
 - DataReaders, 678
- DDS_DataReader_get_qos
 - DataReaders, 688
- DDS_DataReader_get_requested_deadline_missed_status
 - DataReaders, 682
- DDS_DataReader_get_requested_incompatible_qos_status
 - DataReaders, 683
- DDS_DataReader_get_sample_lost_status
 - DataReaders, 685
- DDS_DataReader_get_sample_rejected_status
 - DataReaders, 682
- DDS_DataReader_get_subscriber
 - DataReaders, 681
- DDS_DataReader_get_subscription_matched_status

- DataReaders, 683
- DDS_DataReader_get_topicdescription
 - DataReaders, 681
- DDS_DataReader_is_matched_publication_alive
 - DataReaders, 679
- DDS_DataReader_lookup_topic_query
 - DataReaders, 692
- DDS_DATAREADER_QOS_DEFAULT
 - Subscribers, 600
- DDS_DATAREADER_QOS_PRINT_ALL
 - Subscribers, 600
- DDS_DATAREADER_QOS_USE_TOPIC_QOS
 - Subscribers, 601
- DDS_DataReader_set_listener
 - DataReaders, 690
- DDS_DataReader_set_property
 - DataReaders, 689
- DDS_DataReader_set_qos
 - DataReaders, 687
- DDS_DataReader_set_qos_with_profile
 - DataReaders, 687
- DDS_DataReader_take_discovery_snapshot
 - DataReaders, 693
- DDS_DataReader_wait_for_historical_data
 - DataReaders, 675
- DDS_DataReaderCacheStatus, 1586
 - alive_instance_count, 1590
 - alive_instance_count_peak, 1590
 - compressed_sample_count, 1592
 - content_filter_dropped_sample_count, 1588
 - detached_instance_count, 1592
 - detached_instance_count_peak, 1592
 - disposed_instance_count, 1591
 - disposed_instance_count_peak, 1591
 - expired_dropped_sample_count, 1589
 - no_writers_instance_count, 1591
 - no_writers_instance_count_peak, 1591
 - old_source_timestamp_dropped_sample_count, 1587
 - ownership_dropped_sample_count, 1588
 - replaced_dropped_sample_count, 1589
 - sample_count, 1587
 - sample_count_peak, 1587
 - time_based_filter_dropped_sample_count, 1588
 - tolerance_source_timestamp_dropped_sample_count, 1588
 - total_samples_dropped_by_instance_replacement, 1590
 - virtual_duplicate_dropped_sample_count, 1589
 - writer_removed_batch_sample_dropped_sample_count, 1590
- DDS_DataReaderCacheStatus_copy
 - DataReaders, 661
- DDS_DataReaderCacheStatus_equals
 - DataReaders, 663
- DDS_DataReaderCacheStatus_finalize
 - DataReaders, 662
- DDS_DataReaderCacheStatus_initialize
 - DataReaders, 661
- DDS_DataReaderCacheStatus_INITIALIZER
 - DataReaders, 613
- DDS_DataReaderInstanceRemovalKind
 - DATA_READER_RESOURCE_LIMITS, 1254
- DDS_DataReaderListener, 1593
 - as_listener, 1594
 - on_data_available, 1595
 - on_liveliness_changed, 1594
 - on_requested_deadline_missed, 1594
 - on_requested_incompatible_qos, 1594
 - on_sample_lost, 1595
 - on_sample_rejected, 1594
 - on_subscription_matched, 1595
- DDS_DataReaderListener_DataAvailableCallback
 - DataReaders, 617
- DDS_DataReaderListener_INITIALIZER
 - DataReaders, 615
- DDS_DataReaderListener_LivelinessChangedCallback
 - DataReaders, 617
- DDS_DataReaderListener_RequestedDeadlineMissedCallback
 - DataReaders, 616
- DDS_DataReaderListener_RequestedIncompatibleQosCallback
 - DataReaders, 617
- DDS_DataReaderListener_SampleLostCallback
 - DataReaders, 618
- DDS_DataReaderListener_SampleRejectedCallback
 - DataReaders, 617
- DDS_DataReaderListener_SubscriptionMatchedCallback
 - DataReaders, 617
- DDS_DATAREADERPROTOCOL_QOS_POLICY_ID
 - All QoS Policies, 1248
- DDS_DATAREADERPROTOCOL_QOS_POLICY_NAME
 - DATA_READER_PROTOCOL, 1253
- DDS_DataReaderProtocolQosPolicy, 1596
 - disable_positive_acks, 1598
 - expects_inline_qos, 1597
 - propagate_dispose_of_unregistered_instances, 1598
 - propagate_unregister_of_disposed_instances, 1599
 - rtps_object_id, 1597
 - rtps_reliable_reader, 1599
 - virtual_guid, 1597
- DDS_DataReaderProtocolStatus, 1600
 - dropped_fragment_count, 1610
 - duplicate_sample_bytes, 1604
 - duplicate_sample_bytes_change, 1604
 - duplicate_sample_count, 1604
 - duplicate_sample_count_change, 1604
 - filtered_sample_bytes, 1605

- filtered_sample_bytes_change, 1605
- filtered_sample_count, 1604
- filtered_sample_count_change, 1605
- first_available_sample_sequence_number, 1609
- last_available_sample_sequence_number, 1609
- last_committed_sample_sequence_number, 1609
- out_of_range_rejected_sample_count, 1609
- reassembled_sample_count, 1610
- received_fragment_count, 1610
- received_gap_bytes, 1608
- received_gap_bytes_change, 1608
- received_gap_count, 1607
- received_gap_count_change, 1608
- received_heartbeat_bytes, 1606
- received_heartbeat_bytes_change, 1606
- received_heartbeat_count, 1605
- received_heartbeat_count_change, 1605
- received_sample_bytes, 1603
- received_sample_bytes_change, 1603
- received_sample_count, 1602
- received_sample_count_change, 1603
- rejected_sample_count, 1608
- rejected_sample_count_change, 1608
- sent_ack_bytes, 1606
- sent_ack_bytes_change, 1607
- sent_ack_count, 1606
- sent_ack_count_change, 1606
- sent_nack_bytes, 1607
- sent_nack_bytes_change, 1607
- sent_nack_count, 1607
- sent_nack_count_change, 1607
- sent_nack_fragment_bytes, 1610
- sent_nack_fragment_count, 1610
- uncommitted_sample_count, 1609
- DDS_DataReaderProtocolStatus_copy
DataReaders, 664
- DDS_DataReaderProtocolStatus_equals
DataReaders, 666
- DDS_DataReaderProtocolStatus_finalize
DataReaders, 664
- DDS_DataReaderProtocolStatus_initialize
DataReaders, 663
- DDS_DataReaderProtocolStatus_INITIALIZER
DataReaders, 614
- DDS_DataReaderQos, 1611
 - availability, 1617
 - data_tags, 1615
 - deadline, 1613
 - destination_order, 1614
 - durability, 1613
 - history, 1614
 - latency_budget, 1613
 - liveliness, 1613
 - multicast, 1616
 - ownership, 1614
 - property, 1616
 - protocol, 1616
 - reader_data_lifecycle, 1615
 - reader_resource_limits, 1615
 - reliability, 1614
 - representation, 1615
 - resource_limits, 1614
 - service, 1617
 - subscription_name, 1617
 - time_based_filter, 1615
 - transport_priority, 1617
 - transport_selection, 1616
 - type_consistency, 1615
 - type_support, 1617
 - unicast, 1616
 - user_data, 1614
- DDS_DataReaderQos_copy
DataReaders, 669
- DDS_DataReaderQos_equals
DataReaders, 666
- DDS_DataReaderQos_finalize
DataReaders, 671
- DDS_DataReaderQos_initialize
DataReaders, 669
- DDS_DataReaderQos_INITIALIZER
DataReaders, 614
- DDS_DataReaderQos_print
DataReaders, 667
- DDS_DataReaderQos_to_string
DataReaders, 667
- DDS_DataReaderQos_to_string_w_params
DataReaders, 668
- DDS_DATAREADERRESOURCELIMITS_QOS_POLICY_ID
All QoS Policies, 1247
- DDS_DATAREADERRESOURCELIMITS_QOS_POLICY_NAME
DATA_READER_RESOURCE_LIMITS, 1256
- DDS_DataReaderResourceLimitsInstanceReplacementSettings,
1618
 - alive_instance_removal, 1618
 - disposed_instance_removal, 1618
 - no_writers_instance_removal, 1619
- DDS_DataReaderResourceLimitsQosPolicy, 1619
 - autopurge_remote_not_alive_writer_delay, 1636
 - disable_fragmentation_support, 1626
 - dynamically_allocate_fragmented_samples, 1628
 - initial_fragmented_samples, 1627
 - initial_infos, 1624
 - initial_outstanding_reads, 1625
 - initial_remote_virtual_writers, 1630
 - initial_remote_virtual_writers_per_instance, 1631
 - initial_remote_writers, 1623
 - initial_remote_writers_per_instance, 1624
 - initial_topic_queries, 1634

- instance_replacement, 1635
- keep_minimum_state_for_instances, 1633
- max_app_ack_response_length, 1633
- max_fragmented_samples, 1626
- max_fragmented_samples_per_remote_writer, 1627
- max_fragments_per_sample, 1628
- max_infos, 1623
- max_outstanding_reads, 1625
- max_query_condition_filters, 1632
- max_remote_virtual_writers, 1630
- max_remote_virtual_writers_per_instance, 1631
- max_remote_writers, 1621
- max_remote_writers_per_instance, 1622
- max_remote_writers_per_sample, 1632
- max_samples_per_read, 1625
- max_samples_per_remote_writer, 1622
- max_topic_queries, 1634
- max_total_instances, 1629
- shmem_ref_transfer_mode_attached_segment_allocation, 1635
- DDS_DataReaderSeq, 1637
- DDS_DataReaderStatusConditionHandler
 - AsyncWaitSet, 1075
- DDS_DataReaderStatusConditionHandler_delete
 - AsyncWaitSet, 1089
- DDS_DataReaderStatusConditionHandler_new
 - AsyncWaitSet, 1089
- DDS_DataRepresentationId_t
 - DATA_REPRESENTATION, 1257
- DDS_DataRepresentationIdSeq, 1638
- DDS_DataRepresentationQosPolicy, 1638
 - compression_settings, 1640
 - value, 1640
- DDS_DATATAG_QOS_POLICY_ID
 - All QoS Policies, 1247
- DDS_DATATAG_QOS_POLICY_NAME
 - DATA_TAG, 1268
- DDS_DataTagQosPolicy
 - DATA_TAG, 1264
- DDS_DataTagQosPolicyHelper_add_tag
 - DATA_TAG, 1266
- DDS_DataTagQosPolicyHelper_assert_tag
 - DATA_TAG, 1265
- DDS_DataTagQosPolicyHelper_get_number_of_tags
 - DATA_TAG, 1267
- DDS_DataTagQosPolicyHelper_lookup_tag
 - DATA_TAG, 1265
- DDS_DataTagQosPolicyHelper_remove_tag
 - DATA_TAG, 1266
- DDS_DataTags, 1641
 - tags, 1641
- DDS_DataWriter
 - Data Writers, 485
- DDS_DataWriter_as_entity
 - Data Writers, 537
- DDS_DataWriter_assert_liveliness
 - Data Writers, 538
- DDS_DATAWRITER_C
 - User Data Type Support, 213
- DDS_DataWriter_flush
 - Data Writers, 555
- DDS_DataWriter_get_datawriter_cache_status
 - Data Writers, 548
- DDS_DataWriter_get_datawriter_protocol_status
 - Data Writers, 548
- DDS_DataWriter_get_listener
 - Data Writers, 554
- DDS_DataWriter_get_listenerX
 - Data Writers, 554
- DDS_DataWriter_get_liveliness_lost_status
 - Data Writers, 545
- DDS_DataWriter_get_matched_subscription_data
 - Data Writers, 540
- DDS_DataWriter_get_matched_subscription_datawriter_protocol_status
 - Data Writers, 549
- DDS_DataWriter_get_matched_subscription_datawriter_protocol_status_by
 - Data Writers, 549
- DDS_DataWriter_get_matched_subscription_locators
 - Data Writers, 538
- DDS_DataWriter_get_matched_subscription_participant_data
 - Data Writers, 542
- DDS_DataWriter_get_matched_subscriptions
 - Data Writers, 539
- DDS_DataWriter_get_offered_deadline_missed_status
 - Data Writers, 546
- DDS_DataWriter_get_offered_incompatible_qos_status
 - Data Writers, 546
- DDS_DataWriter_get_publication_matched_status
 - Data Writers, 546
- DDS_DataWriter_get_publisher
 - Data Writers, 543
- DDS_DataWriter_get_qos
 - Data Writers, 552
- DDS_DataWriter_get_reliable_reader_activity_changed_status
 - Data Writers, 547
- DDS_DataWriter_get_reliable_writer_cache_changed_status
 - Data Writers, 547
- DDS_DataWriter_get_service_request_accepted_status
 - Data Writers, 550
- DDS_DataWriter_get_topic
 - Data Writers, 542
- DDS_DataWriter_is_matched_subscription_active
 - Data Writers, 540
- DDS_DataWriter_is_sample_app_acknowledged
 - Data Writers, 544
- DDS_DATAWRITER_QOS_DEFAULT
 - Publishers, 470
- DDS_DATAWRITER_QOS_PRINT_ALL

- Publishers, 469
- DDS_DATAWRITER_QOS_USE_TOPIC_QOS
 - Publishers, 470
- DDS_DataWriter_set_listener
 - Data Writers, 553
- DDS_DataWriter_set_property
 - Data Writers, 552
- DDS_DataWriter_set_qos
 - Data Writers, 550
- DDS_DataWriter_set_qos_with_profile
 - Data Writers, 551
- DDS_DataWriter_take_discovery_snapshot
 - Data Writers, 555
- DDS_DataWriter_wait_for_acknowledgments
 - Data Writers, 543
- DDS_DataWriter_wait_for_asynchronous_publishing
 - Data Writers, 544
- DDS_DataWriterCacheStatus, 1641
 - alive_instance_count, 1643
 - alive_instance_count_peak, 1643
 - disposed_instance_count, 1643
 - disposed_instance_count_peak, 1643
 - sample_count, 1642
 - sample_count_peak, 1642
 - unregistered_instance_count, 1643
 - unregistered_instance_count_peak, 1644
- DDS_DataWriterCacheStatus_copy
 - Data Writers, 529
- DDS_DataWriterCacheStatus_equals
 - Data Writers, 530
- DDS_DataWriterCacheStatus_finalize
 - Data Writers, 529
- DDS_DataWriterCacheStatus_initialize
 - Data Writers, 528
- DDS_DataWriterCacheStatus_INITIALIZER
 - Data Writers, 483
- DDS_DataWriterListener, 1644
 - as_listener, 1645
 - on_application_acknowledgment, 1648
 - on_empty_virtual_writer, 1648
 - on_instance_replaced, 1647
 - on_liveliness_lost, 1646
 - on_offered_deadline_missed, 1645
 - on_offered_incompatible_qos, 1645
 - on_publication_matched, 1646
 - on_reliable_reader_activity_changed, 1646
 - on_reliable_writer_cache_changed, 1646
 - on_sample_removed, 1647
 - on_service_request_accepted, 1648
- DDS_DataWriterListener_INITIALIZER
 - Data Writers, 484
- DDS_DataWriterListener_InstanceReplacedCallback
 - Data Writers, 489
- DDS_DataWriterListener_LivelinessLostCallback
 - Data Writers, 486
- DDS_DataWriterListener_OfferedDeadlineMissedCallback
 - Data Writers, 486
- DDS_DataWriterListener_OfferedIncompatibleQosCallback
 - Data Writers, 487
- DDS_DataWriterListener_OnApplicationAcknowledgmentCallback
 - Data Writers, 489
- DDS_DataWriterListener_PublicationMatchedCallback
 - Data Writers, 487
- DDS_DataWriterListener_ReliableReaderActivityChangedCallback
 - Data Writers, 488
- DDS_DataWriterListener_ReliableWriterCacheChangedCallback
 - Data Writers, 487
- DDS_DataWriterListener_SampleRemovedCallback
 - Data Writers, 488
- DDS_DataWriterListener_ServiceRequestAcceptedCallback
 - Data Writers, 489
- DDS_DATAWRITERPROTOCOL_QOS_POLICY_ID
 - All QoS Policies, 1248
- DDS_DATAWRITERPROTOCOL_QOS_POLICY_NAME
 - DATA_WRITER_PROTOCOL, 1268
- DDS_DataWriterProtocolQosPolicy, 1649
 - disable_inline_keyhash, 1651
 - disable_positive_acks, 1651
 - initial_virtual_sequence_number, 1653
 - propagate_app_ack_with_no_response, 1653
 - push_on_write, 1650
 - rtps_object_id, 1650
 - rtps_reliable_writer, 1653
 - serialize_key_with_dispose, 1652
 - virtual_guid, 1650
- DDS_DataWriterProtocolStatus, 1654
 - filtered_sample_bytes, 1658
 - filtered_sample_bytes_change, 1658
 - filtered_sample_count, 1657
 - filtered_sample_count_change, 1658
 - first_available_sample_sequence_number, 1663
 - first_available_sample_virtual_sequence_number, 1663
 - first_unacknowledged_sample_sequence_number, 1663
 - first_unacknowledged_sample_subscription_handle, 1664
 - first_unacknowledged_sample_virtual_sequence_number, 1664
 - first_unelapsed_keep_duration_sample_sequence_number, 1664
 - last_available_sample_sequence_number, 1663
 - last_available_sample_virtual_sequence_number, 1663
 - pulled_fragment_bytes, 1665
 - pulled_fragment_count, 1665
 - pulled_sample_bytes, 1659
 - pulled_sample_bytes_change, 1660

- pulled_sample_count, 1659
- pulled_sample_count_change, 1659
- pushed_fragment_bytes, 1664
- pushed_fragment_count, 1664
- pushed_sample_bytes, 1657
- pushed_sample_bytes_change, 1657
- pushed_sample_count, 1657
- pushed_sample_count_change, 1657
- received_ack_bytes, 1660
- received_ack_bytes_change, 1660
- received_ack_count, 1660
- received_ack_count_change, 1660
- received_nack_bytes, 1661
- received_nack_bytes_change, 1661
- received_nack_count, 1661
- received_nack_count_change, 1661
- received_nack_fragment_bytes, 1665
- received_nack_fragment_count, 1665
- rejected_sample_count, 1662
- rejected_sample_count_change, 1662
- send_window_size, 1662
- sent_gap_bytes, 1662
- sent_gap_bytes_change, 1662
- sent_gap_count, 1661
- sent_gap_count_change, 1662
- sent_heartbeat_bytes, 1659
- sent_heartbeat_bytes_change, 1659
- sent_heartbeat_count, 1658
- sent_heartbeat_count_change, 1658
- DDS_DataWriterProtocolStatus_copy
 - Data Writers, 531
- DDS_DataWriterProtocolStatus_equals
 - Data Writers, 532
- DDS_DataWriterProtocolStatus_finalize
 - Data Writers, 532
- DDS_DataWriterProtocolStatus_initialize
 - Data Writers, 531
- DDS_DataWriterProtocolStatus_INITIALIZER
 - Data Writers, 483
- DDS_DataWriterQos, 1666
 - availability, 1672
 - batch, 1672
 - data_tags, 1670
 - deadline, 1668
 - destination_order, 1669
 - durability, 1668
 - durability_service, 1668
 - history, 1669
 - latency_budget, 1668
 - lifespan, 1669
 - liveliness, 1668
 - multi_channel, 1672
 - ownership, 1670
 - ownership_strength, 1670
 - property, 1672
 - protocol, 1671
 - publication_name, 1672
 - publish_mode, 1671
 - reliability, 1669
 - representation, 1670
 - resource_limits, 1669
 - service, 1672
 - topic_query_dispatch, 1673
 - transfer_mode, 1673
 - transport_priority, 1669
 - transport_selection, 1671
 - type_support, 1673
 - unicast, 1671
 - user_data, 1670
 - writer_data_lifecycle, 1670
 - writer_resource_limits, 1671
- DDS_DataWriterQos_copy
 - Data Writers, 537
- DDS_DataWriterQos_equals
 - Data Writers, 533
- DDS_DataWriterQos_finalize
 - Data Writers, 536
- DDS_DataWriterQos_initialize
 - Data Writers, 535
- DDS_DataWriterQos_INITIALIZER
 - Data Writers, 484
- DDS_DataWriterQos_print
 - Data Writers, 533
- DDS_DataWriterQos_to_string
 - Data Writers, 534
- DDS_DataWriterQos_to_string_w_params
 - Data Writers, 534
- DDS_DATAWRITERRESOURCELIMITS_QOS_POLICY_ID
 - All QoS Policies, 1247
- DDS_DATAWRITERRESOURCELIMITS_QOS_POLICY_NAME
 - DATA_WRITER_RESOURCE_LIMITS, 1271
- DDS_DataWriterResourceLimitsInstanceReplacementKind
 - DATA_WRITER_RESOURCE_LIMITS, 1269
- DDS_DataWriterResourceLimitsQosPolicy, 1673
 - autoregister_instances, 1678
 - initial_active_topic_queries, 1680
 - initial_batches, 1676
 - initial_concurrent_blocking_threads, 1675
 - initial_virtual_writers, 1679
 - initialize_writer_loaned_sample, 1681
 - instance_replacement, 1677
 - max_active_topic_queries, 1680
 - max_app_ack_remote_readers, 1680
 - max_batches, 1677
 - max_concurrent_blocking_threads, 1675
 - max_remote_reader_filters, 1676
 - max_remote_readers, 1679
 - max_virtual_writers, 1679

- replace_empty_instances, 1678
- writer_loaned_sample_allocation, 1681
- DDS_DataWriterShmemRefTransferModeSettings, 1682
 - enable_data_consistency_check, 1682
- DDS_DATAWRITERTRANSFERMODE_QOS_POLICY_ID
 - All QoS Policies, 1249
- DDS_DATAWRITERTRANSFERMODE_QOS_POLICY_NAME
 - DATA_WRITER_TRANSFER_MODE, 1272
- DDS_DataWriterTransferModeQosPolicy, 1683
 - shmem_ref_settings, 1683
- DDS_DEADLINE_QOS_POLICY_ID
 - All QoS Policies, 1246
- DDS_DEADLINE_QOS_POLICY_NAME
 - DEADLINE, 1273
- DDS_DeadlineQosPolicy, 1684
 - period, 1685
- DDS_DEFAULT_FLOW_CONTROLLER_NAME
 - Flow Controllers, 564
- DDS_DEFAULT_PRINT_FORMAT
 - Topics, 186
- DDS_DELETE_PERSISTENT_JOURNAL
 - DURABILITY, 1286
- DDS_DESTINATIONORDER_QOS_POLICY_ID
 - All QoS Policies, 1247
- DDS_DESTINATIONORDER_QOS_POLICY_NAME
 - DESTINATION_ORDER, 1275
- DDS_DestinationOrderQosPolicy, 1686
 - kind, 1688
 - scope, 1688
 - source_timestamp_tolerance, 1688
- DDS_DestinationOrderQosPolicyKind
 - DESTINATION_ORDER, 1274
- DDS_DestinationOrderQosPolicyScopeKind
 - DESTINATION_ORDER, 1275
- DDS_DISALLOW_TYPE_COERCION
 - TYPE_CONSISTENCY_ENFORCEMENT, 1341
- DDS_DISCOVERY_QOS_POLICY_ID
 - All QoS Policies, 1247
- DDS_DISCOVERY_QOS_POLICY_NAME
 - DISCOVERY, 1276
- DDS_DISCOVERY_SERVICE_SAMPLE
 - Sample Flags, 1388
- DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_ALL
 - DISCOVERY_CONFIG, 1279
- DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_DEFAULT
 - DISCOVERY_CONFIG, 1278
- DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_NONE
 - DISCOVERY_CONFIG, 1278
- DDS_DISCOVERYCONFIG_BUILTIN_DPSE
 - DISCOVERY_CONFIG, 1281
- DDS_DISCOVERYCONFIG_BUILTIN_PLUGIN_MASK_DEFAULT
 - DISCOVERY_CONFIG, 1278
- DDS_DISCOVERYCONFIG_BUILTIN_PLUGIN_MASK_NONE
 - DISCOVERY_CONFIG, 1278
- DDS_DISCOVERYCONFIG_BUILTIN_SDP
 - DISCOVERY_CONFIG, 1280
- DDS_DISCOVERYCONFIG_BUILTIN_SDP2
 - DISCOVERY_CONFIG, 1281
- DDS_DISCOVERYCONFIG_BUILTIN_SEDP
 - DISCOVERY_CONFIG, 1280
- DDS_DISCOVERYCONFIG_BUILTIN_SPDP
 - DISCOVERY_CONFIG, 1280
- DDS_DISCOVERYCONFIG_BUILTIN_SPDP2
 - DISCOVERY_CONFIG, 1281
- DDS_DISCOVERYCONFIG_QOS_POLICY_ID
 - All QoS Policies, 1248
- DDS_DISCOVERYCONFIG_QOS_POLICY_NAME
 - DISCOVERY_CONFIG, 1283
- DDS_DISCOVERYCONFIG_SERVICE_REQUEST_CHANNEL
 - DISCOVERY_CONFIG, 1281
- DDS_DISCOVERYCONFIG_TYPE_LOOKUP_SERVICE_CHANNEL
 - DISCOVERY_CONFIG, 1281
- DDS_DiscoveryConfigBuiltinChannelKind
 - DISCOVERY_CONFIG, 1281
- DDS_DiscoveryConfigBuiltinChannelKindMask
 - DISCOVERY_CONFIG, 1279
- DDS_DiscoveryConfigBuiltinPluginKind
 - DISCOVERY_CONFIG, 1280
- DDS_DiscoveryConfigBuiltinPluginKindMask
 - DISCOVERY_CONFIG, 1279
- DDS_DiscoveryConfigQosPolicy, 1689
 - asynchronous_publisher, 1704
 - builtin_discovery_plugins, 1701
 - default_domain_announcement_period, 1704
 - dns_tracker_polling_period, 1711
 - enabled_builtin_channels, 1701
 - endpoint_type_object_lb_serialization_threshold, 1711
 - ignore_default_domain_announcements, 1705
 - initial_participant_announcements, 1694
 - locator_reachability_assert_period, 1707
 - locator_reachability_change_detection_period, 1708
 - locator_reachability_lease_duration, 1708
 - max_initial_participant_announcement_period, 1695
 - max_liveliness_loss_detection_period, 1694
 - min_initial_participant_announcement_period, 1695
 - new_remote_participant_announcements, 1695
 - participant_announcement_period, 1693
 - participant_configuration_reader, 1714
 - participant_configuration_reader_resource_limits, 1714
 - participant_configuration_writer, 1712
 - participant_configuration_writer_data_lifecycle, 1713
 - participant_configuration_writer_publish_mode, 1712
 - participant_liveliness_assert_period, 1692
 - participant_liveliness_lease_duration, 1692
 - participant_message_reader, 1702
 - participant_message_reader_reliability_kind, 1701

- participant_message_writer, 1702
- participant_reader_resource_limits, 1696
- publication_reader, 1696
- publication_reader_resource_limits, 1697
- publication_writer, 1698
- publication_writer_data_lifecycle, 1699
- publication_writer_publish_mode, 1703
- remote_participant_purge_kind, 1693
- secure_volatile_reader, 1710
- secure_volatile_writer, 1709
- secure_volatile_writer_publish_mode, 1710
- service_request_reader, 1707
- service_request_writer, 1705
- service_request_writer_data_lifecycle, 1706
- service_request_writer_publish_mode, 1706
- subscription_reader, 1697
- subscription_reader_resource_limits, 1698
- subscription_writer, 1700
- subscription_writer_data_lifecycle, 1700
- subscription_writer_publish_mode, 1704
- DDS_DiscoveryQosPolicy, 1715
 - accept_unknown_peers, 1718
 - enable_endpoint_discovery, 1718
 - enabled_transports, 1716
 - initial_peers, 1716
 - metatraffic_transport_priority, 1717
 - multicast_receive_addresses, 1717
- DDS_DISPOSED_INSTANCE_REPLACEMENT
 - DATA_WRITER_RESOURCE_LIMITS, 1271
- DDS_DISPOSED_THEN_ALIVE_INSTANCE_REPLACEMENT
 - DATA_WRITER_RESOURCE_LIMITS, 1271
- DDS_DOMAIN_ID_USE_XML_CONFIG
 - DomainParticipantConfigParams, 1431
- DDS_DomainEntity
 - Entity Support, 1363
- DDS_DomainId_t
 - DomainParticipants, 77
- DDS_DomainParticipant
 - DomainParticipants, 77
- DDS_DomainParticipant_add_peer
 - DomainParticipants, 149
- DDS_DomainParticipant_as_entity
 - DomainParticipants, 83
- DDS_DomainParticipant_assert_liveliness
 - DomainParticipants, 136
- DDS_DomainParticipant_banish_ignored_participants
 - DomainParticipants, 134
- DDS_DomainParticipant_contains_entity
 - DomainParticipants, 139
- DDS_DomainParticipant_create_contentfilteredtopic
 - DomainParticipants, 118
- DDS_DomainParticipant_create_contentfilteredtopic_with_filters
 - DomainParticipants, 119
- DDS_DomainParticipant_create_datareader
 - DomainParticipants, 112
- DDS_DomainParticipant_create_datareader_with_profile
 - DomainParticipants, 113
- DDS_DomainParticipant_create_datawriter
 - DomainParticipants, 108
- DDS_DomainParticipant_create_datawriter_with_profile
 - DomainParticipants, 110
- DDS_DomainParticipant_create_flowcontroller
 - DomainParticipants, 124
- DDS_DomainParticipant_create_multitopic
 - DomainParticipants, 122
- DDS_DomainParticipant_create_publisher
 - DomainParticipants, 103
- DDS_DomainParticipant_create_publisher_with_profile
 - DomainParticipants, 104
- DDS_DomainParticipant_create_subscriber
 - DomainParticipants, 106
- DDS_DomainParticipant_create_subscriber_with_profile
 - DomainParticipants, 107
- DDS_DomainParticipant_create_topic
 - DomainParticipants, 115
- DDS_DomainParticipant_create_topic_with_profile
 - DomainParticipants, 116
- DDS_DomainParticipant_delete_contained_entities
 - DomainParticipants, 135
- DDS_DomainParticipant_delete_contentfilteredtopic
 - DomainParticipants, 119
- DDS_DomainParticipant_delete_datareader
 - DomainParticipants, 114
- DDS_DomainParticipant_delete_datawriter
 - DomainParticipants, 111
- DDS_DomainParticipant_delete_durable_subscription
 - DomainParticipants, 141
- DDS_DomainParticipant_delete_flowcontroller
 - DomainParticipants, 124
- DDS_DomainParticipant_delete_multitopic
 - DomainParticipants, 123
- DDS_DomainParticipant_delete_publisher
 - DomainParticipants, 105
- DDS_DomainParticipant_delete_subscriber
 - DomainParticipants, 108
- DDS_DomainParticipant_delete_topic
 - DomainParticipants, 117
- DDS_DomainParticipant_find_topic
 - DomainParticipants, 126
- DDS_DomainParticipant_get_builtin_subscriber
 - DomainParticipants, 129
- DDS_DomainParticipant_get_current_time
 - DomainParticipants, 138
- DDS_DomainParticipant_get_default_datareader_qos
 - DomainParticipants, 96
- DDS_DomainParticipant_get_default_datawriter_qos
 - DomainParticipants, 91
- DDS_DomainParticipant_get_default_flowcontroller_property
 - DomainParticipants, 112

- DomainParticipants, 98
- DDS_DomainParticipant_get_default_library
 - DomainParticipants, 100
- DDS_DomainParticipant_get_default_profile
 - DomainParticipants, 100
- DDS_DomainParticipant_get_default_profile_library
 - DomainParticipants, 101
- DDS_DomainParticipant_get_default_publisher_qos
 - DomainParticipants, 89
- DDS_DomainParticipant_get_default_subscriber_qos
 - DomainParticipants, 94
- DDS_DomainParticipant_get_default_topic_qos
 - DomainParticipants, 86
- DDS_DomainParticipant_get_discovered_participant_data
 - DomainParticipants, 145
- DDS_DomainParticipant_get_discovered_participant_subject
 - DomainParticipants, 146
- DDS_DomainParticipant_get_discovered_participants
 - DomainParticipants, 144
- DDS_DomainParticipant_get_discovered_participants_from
 - DomainParticipants, 144
- DDS_DomainParticipant_get_discovered_topic_data
 - DomainParticipants, 147
- DDS_DomainParticipant_get_discovered_topics
 - DomainParticipants, 147
- DDS_DomainParticipant_get_dns_tracker_polling_period
 - DomainParticipants, 143
- DDS_DomainParticipant_get_domain_id
 - DomainParticipants, 135
- DDS_DomainParticipant_get_implicit_publisher
 - DomainParticipants, 129
- DDS_DomainParticipant_get_implicit_subscriber
 - DomainParticipants, 130
- DDS_DomainParticipant_get_listener
 - DomainParticipants, 154
- DDS_DomainParticipant_get_listenerX
 - DomainParticipants, 156
- DDS_DomainParticipant_get_participant_protocol_status
 - DomainParticipants, 128
- DDS_DomainParticipant_get_publishers
 - DomainParticipants, 137
- DDS_DomainParticipant_get_qos
 - DomainParticipants, 152
- DDS_DomainParticipant_get_subscribers
 - DomainParticipants, 138
- DDS_DomainParticipant_get_typecode
 - DomainParticipants, 125
- DDS_DomainParticipant_ignore_participant
 - DomainParticipants, 131
- DDS_DomainParticipant_ignore_publication
 - DomainParticipants, 132
- DDS_DomainParticipant_ignore_subscription
 - DomainParticipants, 133
- DDS_DomainParticipant_ignore_topic
 - DomainParticipants, 132
- DDS_DomainParticipant_lookup_contentfilter
 - DomainParticipants, 121
- DDS_DomainParticipant_lookup_datareader_by_name
 - DomainParticipants, 159
- DDS_DomainParticipant_lookup_datawriter_by_name
 - DomainParticipants, 158
- DDS_DomainParticipant_lookup_flowcontroller
 - DomainParticipants, 128
- DDS_DomainParticipant_lookup_publisher_by_name
 - DomainParticipants, 156
- DDS_DomainParticipant_lookup_subscriber_by_name
 - DomainParticipants, 157
- DDS_DomainParticipant_lookup_topicdescription
 - DomainParticipants, 127
- DDS_DOMAINPARTICIPANT_QOS_PRINT_ALL
 - DomainParticipants, 160
- DDS_DomainParticipant_register_contentfilter
 - DomainParticipants, 120
- DDS_DomainParticipant_register_durable_subscription
 - DomainParticipants, 139
- DDS_DomainParticipant_remove_peer
 - DomainParticipants, 150
- DDS_DomainParticipant_resume_endpoint_discovery
 - DomainParticipants, 141
- DDS_DomainParticipant_set_default_datareader_qos
 - DomainParticipants, 97
- DDS_DomainParticipant_set_default_datareader_qos_with_profile
 - DomainParticipants, 97
- DDS_DomainParticipant_set_default_datawriter_qos
 - DomainParticipants, 92
- DDS_DomainParticipant_set_default_datawriter_qos_with_profile
 - DomainParticipants, 93
- DDS_DomainParticipant_set_default_flowcontroller_property
 - DomainParticipants, 99
- DDS_DomainParticipant_set_default_library
 - DomainParticipants, 101
- DDS_DomainParticipant_set_default_profile
 - DomainParticipants, 102
- DDS_DomainParticipant_set_default_publisher_qos
 - DomainParticipants, 89
- DDS_DomainParticipant_set_default_publisher_qos_with_profile
 - DomainParticipants, 90
- DDS_DomainParticipant_set_default_subscriber_qos
 - DomainParticipants, 94
- DDS_DomainParticipant_set_default_subscriber_qos_with_profile
 - DomainParticipants, 95
- DDS_DomainParticipant_set_default_topic_qos
 - DomainParticipants, 87
- DDS_DomainParticipant_set_default_topic_qos_with_profile
 - DomainParticipants, 88
- DDS_DomainParticipant_set_dns_tracker_polling_period
 - DomainParticipants, 142
- DDS_DomainParticipant_set_listener

- DomainParticipants, 154
- DDS_DomainParticipant_set_property
 - DomainParticipants, 153
- DDS_DomainParticipant_set_qos
 - DomainParticipants, 151
- DDS_DomainParticipant_set_qos_with_profile
 - DomainParticipants, 152
- DDS_DomainParticipant_take_discovery_snapshot
 - DomainParticipants, 148
- DDS_DomainParticipant_unregister_contentfilter
 - DomainParticipants, 121
- DDS_DomainParticipantConfigParams_t, 1719
 - domain_entity_qos_library_name, 1720
 - domain_entity_qos_profile_name, 1721
 - domain_id, 1720
 - participant_name, 1720
 - participant_qos_library_name, 1720
 - participant_qos_profile_name, 1720
- DDS_DomainParticipantConfigParams_t_INITIALIZER
 - DomainParticipantConfigParams, 1431
- DDS_DomainParticipantFactory
 - DomainParticipantFactory, 32
- DDS_DomainParticipantFactory_create_participant
 - DomainParticipantFactory, 41
- DDS_DomainParticipantFactory_create_participant_from_config
 - DomainParticipantFactory, 59
- DDS_DomainParticipantFactory_create_participant_from_config_w_qos
 - DomainParticipantFactory, 60
- DDS_DomainParticipantFactory_create_participant_with_profile
 - DomainParticipantFactory, 43
- DDS_DomainParticipantFactory_delete_participant
 - DomainParticipantFactory, 44
- DDS_DomainParticipantFactory_finalize_instance
 - DomainParticipantFactory, 38
- DDS_DomainParticipantFactory_get_datareader_qos_from_profile
 - DomainParticipantFactory, 54
- DDS_DomainParticipantFactory_get_datareader_qos_from_profile_w_qos
 - DomainParticipantFactory, 54
- DDS_DomainParticipantFactory_get_datawriter_qos_from_profile
 - DomainParticipantFactory, 55
- DDS_DomainParticipantFactory_get_datawriter_qos_from_profile_w_qos
 - DomainParticipantFactory, 55
- DDS_DomainParticipantFactory_get_default_library
 - DomainParticipantFactory, 49
- DDS_DomainParticipantFactory_get_default_participant_qos
 - DomainParticipantFactory, 41
- DDS_DomainParticipantFactory_get_default_profile
 - DomainParticipantFactory, 50
- DDS_DomainParticipantFactory_get_default_profile_library
 - DomainParticipantFactory, 51
- DDS_DomainParticipantFactory_get_instance
 - DomainParticipantFactory, 38
- DDS_DomainParticipantFactory_get_participant_factory_qos_from_profile
 - DomainParticipantFactory, 51
- DDS_DomainParticipantFactory_get_participant_qos_from_profile
 - DomainParticipantFactory, 52
- DDS_DomainParticipantFactory_get_participants
 - DomainParticipantFactory, 63
- DDS_DomainParticipantFactory_get_publisher_qos_from_profile
 - DomainParticipantFactory, 52
- DDS_DomainParticipantFactory_get_qos
 - DomainParticipantFactory, 45
- DDS_DomainParticipantFactory_get_qos_profile_libraries
 - DomainParticipantFactory, 57
- DDS_DomainParticipantFactory_get_qos_profiles
 - DomainParticipantFactory, 58
- DDS_DomainParticipantFactory_get_subscriber_qos_from_profile
 - DomainParticipantFactory, 53
- DDS_DomainParticipantFactory_get_topic_qos_from_profile
 - DomainParticipantFactory, 56
- DDS_DomainParticipantFactory_get_topic_qos_from_profile_w_topic_name
 - DomainParticipantFactory, 57
- DDS_DomainParticipantFactory_get_typecode_from_config
 - DomainParticipantFactory, 59
- DDS_DomainParticipantFactory_load_profiles
 - DomainParticipantFactory, 46
- DDS_DomainParticipantFactory_lookup_participant
 - DomainParticipantFactory, 45
- DDS_DomainParticipantFactory_lookup_participant_by_name
 - DomainParticipantFactory, 61
- DDS_DOMAINPARTICIPANTFACTORY_QOS_PRINT_ALL
 - DomainParticipants, 160
- DDS_DomainParticipantFactory_register_type_support
 - DomainParticipantFactory, 62
- DDS_DomainParticipantFactory_RegisterTypeFunction
 - DomainParticipantFactory, 33
- DDS_DomainParticipantFactory_reload_profiles
 - DomainParticipantFactory, 47
- DDS_DomainParticipantFactory_set_default_library
 - DomainParticipantFactory, 48
- DDS_DomainParticipantFactory_set_default_participant_qos
 - DomainParticipantFactory, 39
- DDS_DomainParticipantFactory_set_default_participant_qos_with_profile
 - DomainParticipantFactory, 40
- DDS_DomainParticipantFactory_set_default_profile
 - DomainParticipantFactory, 49
- DDS_DomainParticipantFactory_set_qos
 - DomainParticipantFactory, 46
- DDS_DomainParticipantFactory_set_thread_factory
 - DomainParticipantFactory, 63
- DDS_DomainParticipantFactory_unload_profiles
 - DomainParticipantFactory, 47
- DDS_DomainParticipantFactory_unregister_thread
 - DomainParticipantFactory, 58
- DDS_DomainParticipantFactoryQos, 1721
 - entity_factory, 1722
 - logging, 1722
 - monitoring, 1722

- profile, 1722
- resource_limits, 1722
- DDS_DomainParticipantFactoryQos_copy
 - DomainParticipantFactory, 37
- DDS_DomainParticipantFactoryQos_equals
 - DomainParticipantFactory, 33
- DDS_DomainParticipantFactoryQos_finalize
 - DomainParticipantFactory, 36
- DDS_DomainParticipantFactoryQos_initialize
 - DomainParticipantFactory, 36
- DDS_DomainParticipantFactoryQos_INITIALIZER
 - DomainParticipantFactory, 31
- DDS_DomainParticipantFactoryQos_print
 - DomainParticipantFactory, 34
- DDS_DomainParticipantFactoryQos_to_string
 - DomainParticipantFactory, 34
- DDS_DomainParticipantFactoryQos_to_string_w_params
 - DomainParticipantFactory, 35
- DDS_DomainParticipantListener, 1723
 - as_listener, 1724
 - as_publisherlistener, 1724
 - as_subscriberlistener, 1725
 - as_topiclistener, 1724
 - on_invalid_local_status_advance_notice, 1725
- DDS_DomainParticipantListener_INITIALIZER
 - DomainParticipants, 75
- DDS_DomainParticipantListener_InvalidLocalAdvanceNoticeStatusCallout
 - DomainParticipants, 77
- DDS_DomainParticipantProtocolStatus, 1725
 - corrupted_rtps_message_count, 1726
 - corrupted_rtps_message_count_change, 1726
 - last_corrupted_message_timestamp, 1726
- DDS_DomainParticipantProtocolStatus_copy
 - DomainParticipants, 84
- DDS_DomainParticipantProtocolStatus_equals
 - DomainParticipants, 86
- DDS_DomainParticipantProtocolStatus_finalize
 - DomainParticipants, 85
- DDS_DomainParticipantProtocolStatus_initialize
 - DomainParticipants, 84
- DDS_DomainParticipantProtocolStatus_INITIALIZER
 - DomainParticipants, 76
- DDS_DomainParticipantQos, 1726
 - database, 1730
 - default_unicast, 1729
 - discovery, 1729
 - discovery_config, 1730
 - entity_factory, 1728
 - event, 1729
 - multicast_mapping, 1730
 - participant_name, 1730
 - partition, 1731
 - property, 1730
 - receiver_pool, 1729
 - resource_limits, 1729
 - service, 1730
 - transport_builtin, 1729
 - type_support, 1731
 - user_data, 1728
 - wire_protocol, 1728
- DDS_DomainParticipantQos_copy
 - DomainParticipants, 83
- DDS_DomainParticipantQos_equals
 - DomainParticipants, 79
- DDS_DomainParticipantQos_finalize
 - DomainParticipants, 82
- DDS_DomainParticipantQos_initialize
 - DomainParticipants, 81
- DDS_DomainParticipantQos_INITIALIZER
 - DomainParticipants, 75
- DDS_DomainParticipantQos_print
 - DomainParticipants, 79
- DDS_DomainParticipantQos_to_string
 - DomainParticipants, 80
- DDS_DomainParticipantQos_to_string_w_params
 - DomainParticipants, 81
- DDS_DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_ID
 - All QoS Policies, 1248
- DDS_DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_NAME
 - DOMAIN_PARTICIPANT_RESOURCE_LIMITS,
- DDS_DomainParticipantResourceLimitsIgnoredEntityReplacementKind
 - DOMAIN_PARTICIPANT_RESOURCE_LIMITS,
 - 1284
- DDS_DomainParticipantResourceLimitsQosPolicy, 1731
 - channel_filter_expression_max_length, 1752
 - channel_seq_max_length, 1752
 - content_filter_allocation, 1739
 - content_filter_hash_buckets, 1745
 - content_filtered_topic_allocation, 1739
 - content_filtered_topic_hash_buckets, 1744
 - contentfilter_property_max_length, 1751
 - deserialized_type_object_dynamic_allocation_threshold,
 - 1751
 - flow_controller_allocation, 1740
 - flow_controller_hash_buckets, 1745
 - ignored_entity_allocation, 1738
 - ignored_entity_hash_buckets, 1744
 - ignored_entity_replacement_kind, 1755
 - local_publisher_allocation, 1736
 - local_publisher_hash_buckets, 1741
 - local_reader_allocation, 1735
 - local_reader_hash_buckets, 1741
 - local_subscriber_allocation, 1736
 - local_subscriber_hash_buckets, 1742
 - local_topic_allocation, 1736
 - local_topic_hash_buckets, 1742
 - local_writer_allocation, 1735

- local_writer_hash_buckets, 1741
- matching_reader_writer_pair_allocation, 1738
- matching_reader_writer_pair_hash_buckets, 1744
- matching_writer_reader_pair_allocation, 1738
- matching_writer_reader_pair_hash_buckets, 1743
- max_endpoint_group_cumulative_characters, 1755
- max_endpoint_groups, 1754
- max_gather_destinations, 1745
- max_partition_cumulative_characters, 1748
- max_partitions, 1748
- outstanding_asynchronous_sample_allocation, 1740
- participant_property_list_max_length, 1752
- participant_property_string_max_length, 1753
- participant_user_data_max_length, 1746
- publisher_group_data_max_length, 1746
- query_condition_allocation, 1740
- read_condition_allocation, 1739
- reader_data_tag_list_max_length, 1757
- reader_data_tag_string_max_length, 1758
- reader_property_list_max_length, 1754
- reader_property_string_max_length, 1754
- reader_user_data_max_length, 1747
- remote_participant_allocation, 1737
- remote_participant_hash_buckets, 1743
- remote_reader_allocation, 1737
- remote_reader_hash_buckets, 1743
- remote_topic_query_allocation, 1756
- remote_topic_query_hash_buckets, 1756
- remote_writer_allocation, 1737
- remote_writer_hash_buckets, 1742
- serialized_type_object_dynamic_allocation_threshold, 1750
- shmem_ref_transfer_mode_max_segments, 1758
- subscriber_group_data_max_length, 1747
- topic_data_max_length, 1746
- transport_info_list_max_length, 1755
- type_code_max_serialized_length, 1749
- type_object_max_deserialized_length, 1750
- type_object_max_serialized_length, 1749
- writer_data_tag_list_max_length, 1757
- writer_data_tag_string_max_length, 1757
- writer_property_list_max_length, 1753
- writer_property_string_max_length, 1753
- writer_user_data_max_length, 1747
- DDS_DomainParticipantSeq, 1759
- DDS_Double
 - DDS-Specific Primitive Types, 1203
- DDS_DoubleSeq, 1759
- DDS_DURABILITY_QOS_POLICY_ID
 - All QoS Policies, 1246
- DDS_DURABILITY_QOS_POLICY_NAME
 - DURABILITY, 1288
- DDS_DurabilityQosPolicy, 1759
 - direct_communication, 1762
 - kind, 1762
 - storage_settings, 1763
 - writer_depth, 1762
- DDS_DurabilityQosPolicyKind
 - DURABILITY, 1287
- DDS_DURABILITYSERVICE_QOS_POLICY_ID
 - All QoS Policies, 1247
- DDS_DURABILITYSERVICE_QOS_POLICY_NAME
 - DURABILITY_SERVICE, 1289
- DDS_DurabilityServiceQosPolicy, 1764
 - history_depth, 1765
 - history_kind, 1765
 - max_instances, 1766
 - max_samples, 1766
 - max_samples_per_instance, 1766
 - service_cleanup_delay, 1765
- DDS_DURATION_AUTO
 - Time Support, 1209
- DDS_DURATION_AUTO_NSEC
 - Time Support, 1209
- DDS_DURATION_AUTO_SEC
 - Time Support, 1209
- DDS_DURATION_INFINITE
 - Time Support, 1208
- DDS_DURATION_INFINITE_NSEC
 - Time Support, 1208
- DDS_DURATION_INFINITE_SEC
 - Time Support, 1208
- DDS_Duration_is_auto
 - Time Support, 1207
- DDS_Duration_is_infinite
 - Time Support, 1207
- DDS_Duration_is_zero
 - Time Support, 1207
- DDS_Duration_t, 1767
 - nanosec, 1768
 - sec, 1767
- DDS_DURATION_ZERO
 - Time Support, 1209
- DDS_DURATION_ZERO_NSEC
 - Time Support, 1209
- DDS_DURATION_ZERO_SEC
 - Time Support, 1209
- DDS_DYNAMIC_DATA_JSON_PARSER_PROPERTIES_DEFAULT
 - Dynamic Data, 438
- DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED
 - Dynamic Data, 324
- DDS_DYNAMIC_DATA_PROPERTY_DEFAULT
 - Dynamic Data, 437
- DDS_DYNAMIC_DATA_TYPE_PROPERTY_DEFAULT
 - Dynamic Data, 437
- DDS_DynamicData, 1768
- DDS_DynamicData_bind_complex_member
 - Dynamic Data, 342

- DDS_DynamicData_bind_type
 - Dynamic Data, 340
- DDS_DynamicData_clear_all_members
 - Dynamic Data, 330
- DDS_DynamicData_clear_member
 - Dynamic Data, 331
- DDS_DynamicData_clear_optional_member
 - Dynamic Data, 330
- DDS_DynamicData_copy
 - Dynamic Data, 329
- DDS_DynamicData_delete
 - Dynamic Data, 329
- DDS_DynamicData_equal
 - Dynamic Data, 329
- DDS_DynamicData_finalize
 - Dynamic Data, 328
- DDS_DynamicData_from_cdr_buffer
 - Dynamic Data, 337
- DDS_DynamicData_from_string
 - Dynamic Data, 339
- DDS_DynamicData_get_boolean
 - Dynamic Data, 356
- DDS_DynamicData_get_boolean_array
 - Dynamic Data, 372
- DDS_DynamicData_get_boolean_seq
 - Dynamic Data, 386
- DDS_DynamicData_get_cdr_buffer
 - Dynamic Data, 332
- DDS_DynamicData_get_char
 - Dynamic Data, 356
- DDS_DynamicData_get_char_array
 - Dynamic Data, 373
- DDS_DynamicData_get_char_seq
 - Dynamic Data, 387
- DDS_DynamicData_get_complex_member
 - Dynamic Data, 365
- DDS_DynamicData_get_discriminator
 - Dynamic Data, 335
- DDS_DynamicData_get_double
 - Dynamic Data, 355
- DDS_DynamicData_get_double_array
 - Dynamic Data, 371
- DDS_DynamicData_get_double_seq
 - Dynamic Data, 385
- DDS_DynamicData_get_float
 - Dynamic Data, 354
- DDS_DynamicData_get_float_array
 - Dynamic Data, 370
- DDS_DynamicData_get_float_seq
 - Dynamic Data, 384
- DDS_DynamicData_get_info
 - Dynamic Data, 340
- DDS_DynamicData_get_int8
 - Dynamic Data, 361
- DDS_DynamicData_get_int8_array
 - Dynamic Data, 379
- DDS_DynamicData_get_int8_seq
 - Dynamic Data, 392
- DDS_DynamicData_get_long
 - Dynamic Data, 351
- DDS_DynamicData_get_long_array
 - Dynamic Data, 366
- DDS_DynamicData_get_long_seq
 - Dynamic Data, 381
- DDS_DynamicData_get_longdouble
 - Dynamic Data, 360
- DDS_DynamicData_get_longdouble_array
 - Dynamic Data, 377
- DDS_DynamicData_get_longdouble_seq
 - Dynamic Data, 390
- DDS_DynamicData_get_longlong
 - Dynamic Data, 358
- DDS_DynamicData_get_longlong_array
 - Dynamic Data, 375
- DDS_DynamicData_get_longlong_seq
 - Dynamic Data, 389
- DDS_DynamicData_get_member_count
 - Dynamic Data, 345
- DDS_DynamicData_get_member_info
 - Dynamic Data, 347
- DDS_DynamicData_get_member_info_by_index
 - Dynamic Data, 348
- DDS_DynamicData_get_member_type
 - Dynamic Data, 349
- DDS_DynamicData_get_octet
 - Dynamic Data, 357
- DDS_DynamicData_get_octet_array
 - Dynamic Data, 374
- DDS_DynamicData_get_octet_seq
 - Dynamic Data, 388
- DDS_DynamicData_get_short
 - Dynamic Data, 351
- DDS_DynamicData_get_short_array
 - Dynamic Data, 367
- DDS_DynamicData_get_short_seq
 - Dynamic Data, 382
- DDS_DynamicData_get_string
 - Dynamic Data, 363
- DDS_DynamicData_get_type
 - Dynamic Data, 344
- DDS_DynamicData_get_type_kind
 - Dynamic Data, 345
- DDS_DynamicData_get_uint8
 - Dynamic Data, 362
- DDS_DynamicData_get_uint8_array
 - Dynamic Data, 380
- DDS_DynamicData_get_uint8_seq
 - Dynamic Data, 393

- DDS_DynamicData_get_ulong
 - Dynamic Data, 352
- DDS_DynamicData_get_ulong_array
 - Dynamic Data, 368
- DDS_DynamicData_get_ulong_seq
 - Dynamic Data, 383
- DDS_DynamicData_get_ulonglong
 - Dynamic Data, 359
- DDS_DynamicData_get_ulonglong_array
 - Dynamic Data, 376
- DDS_DynamicData_get_ulonglong_seq
 - Dynamic Data, 389
- DDS_DynamicData_get_ushort
 - Dynamic Data, 353
- DDS_DynamicData_get_ushort_array
 - Dynamic Data, 369
- DDS_DynamicData_get_ushort_seq
 - Dynamic Data, 383
- DDS_DynamicData_get_wchar
 - Dynamic Data, 361
- DDS_DynamicData_get_wchar_array
 - Dynamic Data, 378
- DDS_DynamicData_get_wchar_seq
 - Dynamic Data, 391
- DDS_DynamicData_get_wstring
 - Dynamic Data, 364
- DDS_DynamicData_initialize
 - Dynamic Data, 327
- DDS_DynamicData_is_cdr
 - Dynamic Data, 333
- DDS_DynamicData_is_member_key
 - Dynamic Data, 350
- DDS_DynamicData_member_exists
 - Dynamic Data, 345
- DDS_DynamicData_member_exists_in_type
 - Dynamic Data, 346
- DDS_DynamicData_new
 - Dynamic Data, 327
- DDS_DynamicData_print
 - Dynamic Data, 336
- DDS_DynamicData_set_boolean
 - Dynamic Data, 398
- DDS_DynamicData_set_boolean_array
 - Dynamic Data, 413
- DDS_DynamicData_set_boolean_seq
 - Dynamic Data, 425
- DDS_DynamicData_set_cdr_buffer
 - Dynamic Data, 332
- DDS_DynamicData_set_char
 - Dynamic Data, 399
- DDS_DynamicData_set_char_array
 - Dynamic Data, 414
- DDS_DynamicData_set_char_seq
 - Dynamic Data, 426
- DDS_DynamicData_set_complex_member
 - Dynamic Data, 406
- DDS_DynamicData_set_discriminator
 - Dynamic Data, 334
- DDS_DynamicData_set_double
 - Dynamic Data, 397
- DDS_DynamicData_set_double_array
 - Dynamic Data, 412
- DDS_DynamicData_set_double_seq
 - Dynamic Data, 424
- DDS_DynamicData_set_float
 - Dynamic Data, 396
- DDS_DynamicData_set_float_array
 - Dynamic Data, 411
- DDS_DynamicData_set_float_seq
 - Dynamic Data, 424
- DDS_DynamicData_set_int8
 - Dynamic Data, 403
- DDS_DynamicData_set_int8_array
 - Dynamic Data, 419
- DDS_DynamicData_set_int8_seq
 - Dynamic Data, 430
- DDS_DynamicData_set_long
 - Dynamic Data, 394
- DDS_DynamicData_set_long_array
 - Dynamic Data, 408
- DDS_DynamicData_set_long_seq
 - Dynamic Data, 421
- DDS_DynamicData_set_longdouble
 - Dynamic Data, 402
- DDS_DynamicData_set_longdouble_array
 - Dynamic Data, 417
- DDS_DynamicData_set_longdouble_seq
 - Dynamic Data, 429
- DDS_DynamicData_set_longlong
 - Dynamic Data, 400
- DDS_DynamicData_set_longlong_array
 - Dynamic Data, 415
- DDS_DynamicData_set_longlong_seq
 - Dynamic Data, 427
- DDS_DynamicData_set_octet
 - Dynamic Data, 399
- DDS_DynamicData_set_octet_array
 - Dynamic Data, 415
- DDS_DynamicData_set_octet_seq
 - Dynamic Data, 427
- DDS_DynamicData_set_short
 - Dynamic Data, 394
- DDS_DynamicData_set_short_array
 - Dynamic Data, 408
- DDS_DynamicData_set_short_seq
 - Dynamic Data, 421
- DDS_DynamicData_set_string
 - Dynamic Data, 405

- DDS_DynamicData_set_uint8
 - Dynamic Data, 404
- DDS_DynamicData_set_uint8_array
 - Dynamic Data, 420
- DDS_DynamicData_set_uint8_seq
 - Dynamic Data, 431
- DDS_DynamicData_set_ulong
 - Dynamic Data, 395
- DDS_DynamicData_set_ulong_array
 - Dynamic Data, 409
- DDS_DynamicData_set_ulong_seq
 - Dynamic Data, 422
- DDS_DynamicData_set_ulonglong
 - Dynamic Data, 401
- DDS_DynamicData_set_ulonglong_array
 - Dynamic Data, 416
- DDS_DynamicData_set_ulonglong_seq
 - Dynamic Data, 428
- DDS_DynamicData_set_ushort
 - Dynamic Data, 396
- DDS_DynamicData_set_ushort_array
 - Dynamic Data, 410
- DDS_DynamicData_set_ushort_seq
 - Dynamic Data, 423
- DDS_DynamicData_set_wchar
 - Dynamic Data, 402
- DDS_DynamicData_set_wchar_array
 - Dynamic Data, 418
- DDS_DynamicData_set_wchar_seq
 - Dynamic Data, 430
- DDS_DynamicData_set_wstring
 - Dynamic Data, 405
- DDS_DynamicData_to_cdr_buffer
 - Dynamic Data, 338
- DDS_DynamicData_to_cdr_buffer_ex
 - Dynamic Data, 338
- DDS_DynamicData_to_string
 - Dynamic Data, 339
- DDS_DynamicData_unbind_complex_member
 - Dynamic Data, 344
- DDS_DynamicData_unbind_type
 - Dynamic Data, 341
- DDS_DynamicDataInfo, 1775
 - member_count, 1776
 - stored_size, 1776
- DDS_DynamicDataJsonParserProperties_t, 1776
- DDS_DynamicDataMemberId
 - Dynamic Data, 325
- DDS_DynamicDataMemberInfo, 1777
 - element_count, 1779
 - element_kind, 1779
 - member_exists, 1778
 - member_id, 1777
 - member_kind, 1778
 - member_name, 1778
- DDS_DynamicDataProperty_t, 1779
 - buffer_initial_size, 1780
 - buffer_max_size, 1780
- DDS_DynamicDataProperty_t_INITIALIZER
 - Dynamic Data, 324
- DDS_DynamicDataReader
 - Dynamic Data, 326
- DDS_DynamicDataSeq, 1781
- DDS_DynamicDataTypeProperty_t, 1781
 - data, 1781
 - serialization, 1782
- DDS_DynamicDataTypeProperty_t_INITIALIZER
 - Dynamic Data, 325
- DDS_DynamicDataTypeSerializationProperty_t, 1782
 - max_size_serialized, 1783
 - min_size_serialized, 1783
 - skip_deserialization, 1783
 - trim_to_size, 1783
 - use_42e_compatible_alignment, 1782
- DDS_DynamicDataTypeSupport
 - Dynamic Data, 326
- DDS_DynamicDataTypeSupport_copy_data
 - Dynamic Data, 436
- DDS_DynamicDataTypeSupport_create_data
 - Dynamic Data, 435
- DDS_DynamicDataTypeSupport_delete
 - Dynamic Data, 433
- DDS_DynamicDataTypeSupport_delete_data
 - Dynamic Data, 435
- DDS_DynamicDataTypeSupport_finalize_data
 - Dynamic Data, 436
- DDS_DynamicDataTypeSupport_get_data_type
 - Dynamic Data, 434
- DDS_DynamicDataTypeSupport_get_type_name
 - Dynamic Data, 434
- DDS_DynamicDataTypeSupport_initialize_data
 - Dynamic Data, 436
- DDS_DynamicDataTypeSupport_new
 - Dynamic Data, 432
- DDS_DynamicDataTypeSupport_print_data
 - Dynamic Data, 435
- DDS_DynamicDataTypeSupport_register_type
 - Dynamic Data, 433
- DDS_DynamicDataTypeSupport_unregister_type
 - Dynamic Data, 433
- DDS_DynamicDataWriter
 - Dynamic Data, 325
- DDS_EDF_FLOW_CONTROLLER_SCHED_POLICY
 - Flow Controllers, 560
- DDS_EMPTY_INSTANCE_REMOVAL
 - DATA_READER_RESOURCE_LIMITS, 1255
- DDS_EndpointGroup_t, 1784
 - quorum_count, 1785

- role_name, 1784
- DDS_EndpointGroupSeq, 1785
- DDS_EndpointTrustAlgorithmInfo, 1786
 - Built-in Topic's Trust Types, 312
 - interceptor, 1786
- DDS_EndpointTrustAttributesMask
 - Built-in Topic's Trust Types, 310
- DDS_EndpointTrustInterceptorAlgorithmInfo, 1786
 - Built-in Topic's Trust Types, 312
 - required_mask, 1787
 - supported_mask, 1787
- DDS_EndpointTrustProtectionInfo, 1787
 - bitmask, 1787
 - Built-in Topic's Trust Types, 311
 - plugin_bitmask, 1788
- DDS_Entity
 - Entity Support, 1360
- DDS_Entity_enable
 - Entity Support, 1363
- DDS_Entity_get_entity_kind
 - Entity Support, 1366
- DDS_Entity_get_instance_handle
 - Entity Support, 1366
- DDS_Entity_get_status_changes
 - Entity Support, 1365
- DDS_Entity_get_statuscondition
 - Entity Support, 1365
- DDS_ENTITY_NAME_USE_XML_CONFIG
 - DomainParticipantConfigParams, 1431
- DDS_ENTITYFACTORY_QOS_POLICY_ID
 - All QoS Policies, 1247
- DDS_ENTITYFACTORY_QOS_POLICY_NAME
 - ENTITY_FACTORY, 1290
- DDS_EntityFactoryQosPolicy, 1788
 - autoenable_created_entities, 1789
- DDS_ENTITYNAME_QOS_POLICY_ID
 - All QoS Policies, 1248
- DDS_ENTITYNAME_QOS_POLICY_NAME
 - ENTITY_NAME, 1291
- DDS_EntityNameQosPolicy, 1790
 - name, 1790
 - role_name, 1791
- DDS_Enum
 - DDS-Specific Primitive Types, 1204
- DDS_EnumMember, 1791
 - name, 1792
 - ordinal, 1792
- DDS_EnumMemberSeq, 1792
- DDS_EVENT_QOS_POLICY_ID
 - All QoS Policies, 1248
- DDS_EVENT_QOS_POLICY_NAME
 - EVENT, 1292
- DDS_EventQosPolicy, 1793
 - initial_count, 1794
 - max_count, 1794
 - thread, 1793
- DDS_ExceptionCode_t
 - Exception Codes, 1219
- DDS_EXCLUSIVE_OWNERSHIP_QOS
 - OWNERSHIP, 1302
- DDS_ExpressionProperty, 1795
 - key_only_filter, 1795
 - writer_side_filter_optimization, 1795
- DDS_ExtensibilityKind
 - Type Code Support, 243
- DDS_EXTENSIBLE_EXTENSIBILITY
 - Type Code Support, 244
- DDS_FilterSampleInfo, 1796
 - related_reader_guid, 1797
 - related_sample_identity, 1796
 - related_source_guid, 1797
 - sample_identity, 1797
 - source_guid, 1798
- DDS_FILTERSAMPLEINFO_DEFAULT
 - Topics, 209
- DDS_FINAL_EXTENSIBILITY
 - Type Code Support, 244
- DDS_FIXED_RATE_FLOW_CONTROLLER_NAME
 - Flow Controllers, 564
- DDS_Float
 - DDS-Specific Primitive Types, 1203
- DDS_FloatSeq, 1798
- DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT
 - DomainParticipants, 162
- DDS_FlowController
 - Flow Controllers, 559
- DDS_FlowController_get_name
 - Flow Controllers, 561
- DDS_FlowController_get_participant
 - Flow Controllers, 561
- DDS_FlowController_get_property
 - Flow Controllers, 563
- DDS_FlowController_set_property
 - Flow Controllers, 562
- DDS_FlowController_trigger_flow
 - Flow Controllers, 563
- DDS_FlowControllerProperty_t, 1799
 - scheduling_policy, 1799
 - token_bucket, 1800
- DDS_FlowControllerSchedulingPolicy
 - Flow Controllers, 559
- DDS_FlowControllerTokenBucketProperty_t, 1800
 - bytes_per_token, 1802
 - max_tokens, 1801
 - period, 1802
 - tokens_added_per_period, 1801
 - tokens_leaked_per_period, 1802
- DDS_FlowControllerTokenBucketProperty_t_INITIALIZER

- Flow Controllers, 558
- DDS_FULL_PERSISTENT_SYNCHRONIZATION
 - DURABILITY, 1287
- DDS_FULLY_PROCESSED_INSTANCE_REMOVAL
 - DATA_READER_RESOURCE_LIMITS, 1255
- DDS_g_tc_boolean
 - Type Code Support, 302
- DDS_g_tc_char
 - Type Code Support, 303
- DDS_g_tc_double
 - Type Code Support, 302
- DDS_g_tc_float
 - Type Code Support, 302
- DDS_g_tc_int8
 - Type Code Support, 303
- DDS_g_tc_long
 - Type Code Support, 301
- DDS_g_tc_longdouble
 - Type Code Support, 305
- DDS_g_tc_longlong
 - Type Code Support, 304
- DDS_g_tc_null
 - Type Code Support, 300
- DDS_g_tc_octet
 - Type Code Support, 303
- DDS_g_tc_short
 - Type Code Support, 300
- DDS_g_tc_uint8
 - Type Code Support, 304
- DDS_g_tc_ulong
 - Type Code Support, 301
- DDS_g_tc_ulonglong
 - Type Code Support, 304
- DDS_g_tc_ushort
 - Type Code Support, 301
- DDS_g_tc_wchar
 - Type Code Support, 305
- DDS_GROUP_PRESENTATION_QOS
 - Presentation, 1305
- DDS_GROUPDATA_QOS_POLICY_ID
 - All QoS Policies, 1247
- DDS_GROUPDATA_QOS_POLICY_NAME
 - GROUP_DATA, 1294
- DDS_GroupDataQosPolicy, 1803
 - value, 1804
- DDS_GuardCondition
 - Conditions and WaitSets, 1370
- DDS_GuardCondition_as_condition
 - Conditions and WaitSets, 1376
- DDS_GuardCondition_delete
 - Conditions and WaitSets, 1376
- DDS_GuardCondition_new
 - Conditions and WaitSets, 1376
- DDS_GuardCondition_set_trigger_value
 - Conditions and WaitSets, 1377
- DDS_GUID_AUTO
 - GUID Support, 1213
- DDS_GUID_compare
 - GUID Support, 1212
- DDS_GUID_copy
 - GUID Support, 1213
- DDS_GUID_equals
 - GUID Support, 1212
- DDS_GUID_t, 1804
 - GUID Support, 1211
 - value, 1805
- DDS_GUID_UNKNOWN
 - GUID Support, 1213
- DDS_GUID_ZERO
 - GUID Support, 1213
- DDS_HANDLE_NIL
 - User Data Type Support, 228
- DDS_Heap_calloc
 - Heap Support in C, 1393
- DDS_Heap_free
 - Heap Support in C, 1393
- DDS_Heap_malloc
 - Heap Support in C, 1393
- DDS_HIGHEST_OFFERED_PRESENTATION_QOS
 - Presentation, 1305
- DDS_HISTORY_QOS_POLICY_ID
 - All QoS Policies, 1247
- DDS_HISTORY_QOS_POLICY_NAME
 - HISTORY, 1293
- DDS_HistoryQosPolicy, 1805
 - depth, 1807
 - kind, 1807
- DDS_HistoryQosPolicyKind
 - HISTORY, 1292
- DDS_HPF_FLOW_CONTROLLER_SCHED_POLICY
 - Flow Controllers, 561
- DDS_IMMUTABLE_TYPECODE_SYSTEM_EXCEPTION_CODE
 - Exception Codes, 1220
- DDS_INCOMPLETE_SNAPSHOT_TOPIC_QUERY
 - Sample Flags, 1388
- DDS_INCONSISTENT_TOPIC_STATUS
 - Status Kinds, 1228
- DDS_InconsistentTopicStatus, 1808
 - total_count, 1809
 - total_count_change, 1809
- DDS_InconsistentTopicStatus_copy
 - Topics, 187
- DDS_InconsistentTopicStatus_equals
 - Topics, 188
- DDS_InconsistentTopicStatus_finalize
 - Topics, 188
- DDS_InconsistentTopicStatus_initialize
 - Topics, 186

- DDS_InconsistentTopicStatus_INITIALIZER
 - Topics, 173
- DDS_INSTANCE_PRESENTATION_QOS
 - PRESENTATION, 1305
- DDS_INSTANCE_SCOPE_DESTINATIONORDER_QOS
 - DESTINATION_ORDER, 1275
- DDS_INSTANCE_STATE_SERVICE_REQUEST_ID
 - ServiceRequest Built-in Topic, 1101
- DDS_InstanceHandle_compare
 - User Data Type Support, 227
- DDS_InstanceHandle_copy
 - User Data Type Support, 227
- DDS_InstanceHandle_equals
 - User Data Type Support, 226
- DDS_InstanceHandle_is_nil
 - User Data Type Support, 228
- DDS_InstanceHandle_t
 - User Data Type Support, 214
- DDS_InstanceHandleSeq, 1809
- DDS_InstanceStateConsistencyKind
 - RELIABILITY, 1324
- DDS_InstanceStateKind
 - Instance States, 714
- DDS_InstanceStateMask
 - Instance States, 714
- DDS_Int8
 - DDS-Specific Primitive Types, 1202
- DDS_Int8Seq, 1810
- DDS_INTERMEDIATE_REPLY_SEQUENCE_SAMPLE
 - Sample Flags, 1388
- DDS_INTERMEDIATE_TOPIC_QUERY_SAMPLE
 - Sample Flags, 1388
- DDS_INTEROPERABLE_RTPS_WELL_KNOWN_PORTS
 - WIRE_PROTOCOL, 1349
- DDS_INVALID_LOCAL_ADVANCE_NOTICE_STATUS
 - Status Kinds, 1232
- DDS_INVALID_LOCAL_ADVANCE_NOTICE_STATUS_EXPIRATION_CODE
 - DomainParticipants, 75
- DDS_INVALID_QOS_POLICY_ID
 - All QoS Policies, 1246
- DDS_InvalidLocalAdvanceNoticeStatus, 1810
 - expiration_code, 1811
 - expiration_message, 1811
 - expiration_time, 1811
- DDS_InvalidLocalAdvanceNoticeStatus_equals
 - DomainParticipants, 79
- DDS_JSON_PRINT_FORMAT
 - Topics, 186
- DDS_KEEP_ALL_HISTORY_QOS
 - HISTORY, 1293
- DDS_KEEP_LAST_HISTORY_QOS
 - HISTORY, 1293
- DDS_KeyedOctets, 1811
 - KeyedOctets Built-in Type, 1174
- DDS_KeyedOctets_delete
 - KeyedOctets Built-in Type, 1176
- DDS_KeyedOctets_new
 - KeyedOctets Built-in Type, 1175
- DDS_KeyedOctets_new_w_size
 - KeyedOctets Built-in Type, 1176
- DDS_KeyedOctetsDataReader
 - KeyedOctets Built-in Type, 1175
- DDS_KeyedOctetsDataReader_as_datareader
 - KeyedOctets Built-in Type, 1192
- DDS_KeyedOctetsDataReader_get_key_value
 - KeyedOctets Built-in Type, 1197
- DDS_KeyedOctetsDataReader_get_key_value_w_key
 - KeyedOctets Built-in Type, 1198
- DDS_KeyedOctetsDataReader_lookup_instance
 - KeyedOctets Built-in Type, 1198
- DDS_KeyedOctetsDataReader_lookup_instance_w_key
 - KeyedOctets Built-in Type, 1198
- DDS_KeyedOctetsDataReader_narrow
 - KeyedOctets Built-in Type, 1192
- DDS_KeyedOctetsDataReader_read
 - KeyedOctets Built-in Type, 1192
- DDS_KeyedOctetsDataReader_read_instance
 - KeyedOctets Built-in Type, 1194
- DDS_KeyedOctetsDataReader_read_instance_w_condition
 - KeyedOctets Built-in Type, 1195
- DDS_KeyedOctetsDataReader_read_next_instance
 - KeyedOctets Built-in Type, 1196
- DDS_KeyedOctetsDataReader_read_next_instance_w_condition
 - KeyedOctets Built-in Type, 1196
- DDS_KeyedOctetsDataReader_read_next_sample
 - KeyedOctets Built-in Type, 1194
- DDS_KeyedOctetsDataReader_read_w_condition
 - KeyedOctets Built-in Type, 1193
- DDS_KeyedOctetsDataReader_return_loan
 - KeyedOctets Built-in Type, 1197
- DDS_KeyedOctetsDataReader_take_instance
 - KeyedOctets Built-in Type, 1193
- DDS_KeyedOctetsDataReader_take_instance_w_condition
 - KeyedOctets Built-in Type, 1195
- DDS_KeyedOctetsDataReader_take_next_instance
 - KeyedOctets Built-in Type, 1196
- DDS_KeyedOctetsDataReader_take_next_instance_w_condition
 - KeyedOctets Built-in Type, 1197
- DDS_KeyedOctetsDataReader_take_next_sample
 - KeyedOctets Built-in Type, 1194
- DDS_KeyedOctetsDataReader_take_w_condition
 - KeyedOctets Built-in Type, 1193
- DDS_KeyedOctetsDataWriter
 - KeyedOctets Built-in Type, 1175
- DDS_KeyedOctetsDataWriter_as_datawriter
 - KeyedOctets Built-in Type, 1181

- DDS_KeyedOctetsDataWriter_create_data
KeyedOctets Built-in Type, 1184
- DDS_KeyedOctetsDataWriter_delete_data
KeyedOctets Built-in Type, 1184
- DDS_KeyedOctetsDataWriter_dispose
KeyedOctets Built-in Type, 1189
- DDS_KeyedOctetsDataWriter_dispose_w_key
KeyedOctets Built-in Type, 1189
- DDS_KeyedOctetsDataWriter_dispose_w_key_w_timestamp
KeyedOctets Built-in Type, 1190
- DDS_KeyedOctetsDataWriter_dispose_w_timestamp
KeyedOctets Built-in Type, 1190
- DDS_KeyedOctetsDataWriter_get_key_value
KeyedOctets Built-in Type, 1190
- DDS_KeyedOctetsDataWriter_get_key_value_w_key
KeyedOctets Built-in Type, 1191
- DDS_KeyedOctetsDataWriter_lookup_instance
KeyedOctets Built-in Type, 1191
- DDS_KeyedOctetsDataWriter_lookup_instance_w_key
KeyedOctets Built-in Type, 1191
- DDS_KeyedOctetsDataWriter_narrow
KeyedOctets Built-in Type, 1181
- DDS_KeyedOctetsDataWriter_register_instance
KeyedOctets Built-in Type, 1181
- DDS_KeyedOctetsDataWriter_register_instance_w_key
KeyedOctets Built-in Type, 1182
- DDS_KeyedOctetsDataWriter_register_instance_w_key_w_timestamp
KeyedOctets Built-in Type, 1182
- DDS_KeyedOctetsDataWriter_register_instance_w_timestamp
KeyedOctets Built-in Type, 1182
- DDS_KeyedOctetsDataWriter_unregister_instance
KeyedOctets Built-in Type, 1183
- DDS_KeyedOctetsDataWriter_unregister_instance_w_key
KeyedOctets Built-in Type, 1183
- DDS_KeyedOctetsDataWriter_unregister_instance_w_key_w_timestamp
KeyedOctets Built-in Type, 1184
- DDS_KeyedOctetsDataWriter_unregister_instance_w_timestamp
KeyedOctets Built-in Type, 1183
- DDS_KeyedOctetsDataWriter_write
KeyedOctets Built-in Type, 1185
- DDS_KeyedOctetsDataWriter_write_octets_seq_w_key
KeyedOctets Built-in Type, 1186
- DDS_KeyedOctetsDataWriter_write_octets_seq_w_key_w_params
KeyedOctets Built-in Type, 1189
- DDS_KeyedOctetsDataWriter_write_octets_seq_w_key_w_timestamp
KeyedOctets Built-in Type, 1187
- DDS_KeyedOctetsDataWriter_write_octets_w_key
KeyedOctets Built-in Type, 1185
- DDS_KeyedOctetsDataWriter_write_octets_w_key_w_params
KeyedOctets Built-in Type, 1188
- DDS_KeyedOctetsDataWriter_write_octets_w_key_w_timestamp
KeyedOctets Built-in Type, 1186
- DDS_KeyedOctetsDataWriter_write_w_params
KeyedOctets Built-in Type, 1188
- DDS_KeyedOctetsDataWriter_write_w_timestamp
KeyedOctets Built-in Type, 1186
- DDS_KeyedOctetsSeq, 1812
- DDS_KeyedOctetsTypeSupport, 1812
- DDS_KeyedOctetsTypeSupport_data_to_string
KeyedOctets Built-in Type, 1180
- DDS_KeyedOctetsTypeSupport_deserialize_data_from_cdr_buffer
KeyedOctets Built-in Type, 1180
- DDS_KeyedOctetsTypeSupport_get_type_name
KeyedOctets Built-in Type, 1178
- DDS_KeyedOctetsTypeSupport_get_typecode
KeyedOctets Built-in Type, 1179
- DDS_KeyedOctetsTypeSupport_print_data
KeyedOctets Built-in Type, 1179
- DDS_KeyedOctetsTypeSupport_register_type
KeyedOctets Built-in Type, 1176
- DDS_KeyedOctetsTypeSupport_serialize_data_to_cdr_buffer
KeyedOctets Built-in Type, 1179
- DDS_KeyedOctetsTypeSupport_serialize_data_to_cdr_buffer_ex
KeyedOctets Built-in Type, 1180
- DDS_KeyedOctetsTypeSupport_unregister_type
KeyedOctets Built-in Type, 1177
- DDS_KeyedString, 1812
- KeyedString Built-in Type, 1129
- DDS_KeyedString_delete
KeyedString Built-in Type, 1130
- DDS_KeyedString_new
KeyedString Built-in Type, 1129
- DDS_KeyedString_new_w_size
KeyedString Built-in Type, 1130
- DDS_KeyedStringDataReader
KeyedString Built-in Type, 1129
- DDS_KeyedStringDataReader_as_datareader
KeyedString Built-in Type, 1144
- DDS_KeyedStringDataReader_get_key_value
KeyedString Built-in Type, 1149
- DDS_KeyedStringDataReader_get_key_value_w_key
KeyedString Built-in Type, 1149
- DDS_KeyedStringDataReader_lookup_instance
KeyedString Built-in Type, 1150
- DDS_KeyedStringDataReader_lookup_instance_w_key
KeyedString Built-in Type, 1150
- DDS_KeyedStringDataReader_narrow
KeyedString Built-in Type, 1143
- DDS_KeyedStringDataReader_read
KeyedString Built-in Type, 1144
- DDS_KeyedStringDataReader_read_instance
KeyedString Built-in Type, 1146
- DDS_KeyedStringDataReader_read_instance_w_condition
KeyedString Built-in Type, 1147
- DDS_KeyedStringDataReader_read_next_instance
KeyedString Built-in Type, 1147
- DDS_KeyedStringDataReader_read_next_instance_w_condition
KeyedString Built-in Type, 1148

- DDS_KeyedStringDataReader_read_next_sample
KeyedString Built-in Type, 1145
- DDS_KeyedStringDataReader_read_w_condition
KeyedString Built-in Type, 1145
- DDS_KeyedStringDataReader_return_loan
KeyedString Built-in Type, 1149
- DDS_KeyedStringDataReader_take
KeyedString Built-in Type, 1144
- DDS_KeyedStringDataReader_take_instance
KeyedString Built-in Type, 1146
- DDS_KeyedStringDataReader_take_instance_w_condition
KeyedString Built-in Type, 1147
- DDS_KeyedStringDataReader_take_next_instance
KeyedString Built-in Type, 1148
- DDS_KeyedStringDataReader_take_next_instance_w_condition
KeyedString Built-in Type, 1148
- DDS_KeyedStringDataReader_take_next_sample
KeyedString Built-in Type, 1146
- DDS_KeyedStringDataReader_take_w_condition
KeyedString Built-in Type, 1145
- DDS_KeyedStringDataWriter
KeyedString Built-in Type, 1129
- DDS_KeyedStringDataWriter_as_datawriter
KeyedString Built-in Type, 1135
- DDS_KeyedStringDataWriter_create_data
KeyedString Built-in Type, 1138
- DDS_KeyedStringDataWriter_delete_data
KeyedString Built-in Type, 1138
- DDS_KeyedStringDataWriter_dispose
KeyedString Built-in Type, 1141
- DDS_KeyedStringDataWriter_dispose_w_key
KeyedString Built-in Type, 1141
- DDS_KeyedStringDataWriter_dispose_w_key_w_timestamp
KeyedString Built-in Type, 1142
- DDS_KeyedStringDataWriter_dispose_w_timestamp
KeyedString Built-in Type, 1141
- DDS_KeyedStringDataWriter_get_key_value
KeyedString Built-in Type, 1142
- DDS_KeyedStringDataWriter_get_key_value_w_key
KeyedString Built-in Type, 1142
- DDS_KeyedStringDataWriter_lookup_instance
KeyedString Built-in Type, 1143
- DDS_KeyedStringDataWriter_lookup_instance_w_key
KeyedString Built-in Type, 1143
- DDS_KeyedStringDataWriter_narrow
KeyedString Built-in Type, 1135
- DDS_KeyedStringDataWriter_register_instance
KeyedString Built-in Type, 1135
- DDS_KeyedStringDataWriter_register_instance_w_key
KeyedString Built-in Type, 1136
- DDS_KeyedStringDataWriter_register_instance_w_key_w_timestamp
KeyedString Built-in Type, 1136
- DDS_KeyedStringDataWriter_register_instance_w_timestamp
KeyedString Built-in Type, 1136
- DDS_KeyedStringDataWriter_unregister_instance
KeyedString Built-in Type, 1137
- DDS_KeyedStringDataWriter_unregister_instance_w_key
KeyedString Built-in Type, 1137
- DDS_KeyedStringDataWriter_unregister_instance_w_key_w_timestamp
KeyedString Built-in Type, 1138
- DDS_KeyedStringDataWriter_unregister_instance_w_timestamp
KeyedString Built-in Type, 1137
- DDS_KeyedStringDataWriter_write
KeyedString Built-in Type, 1139
- DDS_KeyedStringDataWriter_write_string_w_key
KeyedString Built-in Type, 1139
- DDS_KeyedStringDataWriter_write_string_w_key_w_params
KeyedString Built-in Type, 1140
- DDS_KeyedStringDataWriter_write_string_w_key_w_timestamp
KeyedString Built-in Type, 1140
- DDS_KeyedStringDataWriter_write_w_params
KeyedString Built-in Type, 1140
- DDS_KeyedStringDataWriter_write_w_timestamp
KeyedString Built-in Type, 1139
- DDS_KeyedStringSeq, 1813
- DDS_KeyedStringTypeSupport, 1813
- DDS_KeyedStringTypeSupport_data_to_string
KeyedString Built-in Type, 1134
- DDS_KeyedStringTypeSupport_deserialize_data_from_cdr_buffer
KeyedString Built-in Type, 1134
- DDS_KeyedStringTypeSupport_get_type_name
KeyedString Built-in Type, 1132
- DDS_KeyedStringTypeSupport_get_typecode
KeyedString Built-in Type, 1133
- DDS_KeyedStringTypeSupport_print_data
KeyedString Built-in Type, 1133
- DDS_KeyedStringTypeSupport_register_type
KeyedString Built-in Type, 1130
- DDS_KeyedStringTypeSupport_serialize_data_to_cdr_buffer
KeyedString Built-in Type, 1133
- DDS_KeyedStringTypeSupport_serialize_data_to_cdr_buffer_ex
KeyedString Built-in Type, 1134
- DDS_KeyedStringTypeSupport_unregister_type
KeyedString Built-in Type, 1131
- DDS_LAST_SHARED_READER_QUEUE_SAMPLE
Sample Flags, 1388
- DDS_LATENCYBUDGET_QOS_POLICY_ID
All QoS Policies, 1246
- DDS_LATENCYBUDGET_QOS_POLICY_NAME
LATENCY_BUDGET, 1295
- DDS_LatencyBudgetQosPolicy, 1813
duration, 1815
- DDS_LENGTH_AUTO
RECEIVER_POOL, 1321
- DDS_LENGTH_UNLIMITED
RECEIVER_POOL, 1321
- DDS_RESOURCE_LIMITS, 1326
- DDS_LIFESPAN_QOS_POLICY_ID
All QoS Policies, 1247

- DDS_LIFESPAN_QOS_POLICY_NAME
 - LIFESPAN, 1295
- DDS_LifespanQosPolicy, 1815
 - duration, 1816
- DDS_Listener, 1816
 - listener_data, 1819
- DDS_Listener_INITIALIZER
 - Entity Support, 1360
- DDS_LIVE_STREAM
 - Stream Kinds, 716
- DDS_LIVELINESS_BASED_REMOTE_PARTICIPANT_PURGE
 - Common types and functions, 1111
- DISCOVERY_CONFIG, 1282
- DDS_LIVELINESS_CHANGED_STATUS
 - Status Kinds, 1231
- DDS_LIVELINESS_LOST_STATUS
 - Status Kinds, 1231
- DDS_LIVELINESS_QOS_POLICY_ID
 - All QoS Policies, 1247
- DDS_LIVELINESS_QOS_POLICY_NAME
 - LIVELINESS, 1297
- DDS_LivelinessChangedStatus, 1820
 - alive_count, 1821
 - alive_count_change, 1821
 - last_publication_handle, 1821
 - not_alive_count, 1821
 - not_alive_count_change, 1821
- DDS_LivelinessChangedStatus_copy
 - DataReaders, 652
- DDS_LivelinessChangedStatus_equals
 - DataReaders, 653
- DDS_LivelinessChangedStatus_finalize
 - DataReaders, 652
- DDS_LivelinessChangedStatus_initialize
 - DataReaders, 651
- DDS_LivelinessChangedStatus_INITIALIZER
 - DataReaders, 611
- DDS_LivelinessLostStatus, 1822
 - total_count, 1822
 - total_count_change, 1823
- DDS_LivelinessLostStatus_copy
 - Data Writers, 513
- DDS_LivelinessLostStatus_equals
 - Data Writers, 515
- DDS_LivelinessLostStatus_finalize
 - Data Writers, 514
- DDS_LivelinessLostStatus_initialize
 - Data Writers, 513
- DDS_LivelinessLostStatus_INITIALIZER
 - Data Writers, 480
- DDS_LivelinessQosPolicy, 1823
 - assertions_per_lease_duration, 1826
 - kind, 1825
 - lease_duration, 1825
- DDS_LivelinessQosPolicyKind
 - LIVELINESS, 1296
- DDS_LOCATOR_ADDRESS_INVALID
 - Common types and functions, 1110
- DDS_LOCATOR_ADDRESS_LENGTH_MAX
 - Common types and functions, 1104
- DDS_LOCATOR_INVALID
 - Common types and functions, 1109
- DDS_LOCATOR_KIND_INVALID
 - Common types and functions, 1109
- DDS_LOCATOR_KIND_RESERVED
 - Common types and functions, 1111
- DDS_LOCATOR_KIND_SHMEM
 - Common types and functions, 1110
- DDS_LOCATOR_KIND_SHMEM_510
 - Common types and functions, 1110
- DDS_LOCATOR_KIND_UDPv4
 - Common types and functions, 1110
- DDS_LOCATOR_KIND_UDPv4_WAN
 - Common types and functions, 1110
- DDS_LOCATOR_KIND_UDPv6
 - Common types and functions, 1110
- DDS_LOCATOR_KIND_UDPv6_510
 - Common types and functions, 1111
- DDS_LOCATOR_PORT_INVALID
 - Common types and functions, 1109
- DDS_LOCATOR_REACHABILITY_SERVICE_REQUEST_ID
 - ServiceRequest Built-in Topic, 1101
- DDS_Locator_t, 1826
 - address, 1827
 - Common types and functions, 1106
 - kind, 1827
 - port, 1827
- DDS_LOCATORFILTER_QOS_POLICY_ID
 - All QoS Policies, 1248
- DDS_LOCATORFILTER_QOS_POLICY_NAME
 - LOCATORFILTER, 1298
- DDS_LocatorFilter_t, 1828
 - filter_expression, 1828
 - locators, 1828
- DDS_LocatorFilterQosPolicy, 1829
 - filter_name, 1829
 - locator_filters, 1829
- DDS_LocatorFilterSeq, 1830
- DDS_LocatorSeq, 1830
- DDS_LOGGING_QOS_POLICY_ID
 - All QoS Policies, 1249
- DDS_LoggingQosPolicy, 1831
 - category, 1832
 - max_bytes_per_file, 1833
 - max_files, 1834
 - output_file, 1832
 - output_file_suffix, 1833
 - print_format, 1832
 - verbosity, 1832

- DDS_Long
 - DDS-Specific Primitive Types, 1203
- DDS_LongDouble
 - DDS-Specific Primitive Types, 1204
- DDS_LongDoubleSeq, 1835
- DDS_LongLong
 - DDS-Specific Primitive Types, 1203
- DDS_LongLongSeq, 1835
- DDS_LongSeq, 1835
- DDS_LOST_BY_AVAILABILITY_WAITING_TIME
 - DataReaders, 620
- DDS_LOST_BY_DECODE_FAILURE
 - DataReaders, 621
- DDS_LOST_BY_DESERIALIZATION_FAILURE
 - DataReaders, 621
- DDS_LOST_BY_INCOMPLETE_COHERENT_SET
 - DataReaders, 619
- DDS_LOST_BY_INSTANCES_LIMIT
 - DataReaders, 618
- DDS_LOST_BY_LARGE_COHERENT_SET
 - DataReaders, 619
- DDS_LOST_BY_OUT_OF_MEMORY
 - DataReaders, 621
- DDS_LOST_BY_REMOTE_WRITER_SAMPLES_PER_VIRTUAL_INSTANCE_LIMIT
 - DataReaders, 621
- DDS_LOST_BY_REMOTE_WRITERS_PER_INSTANCE_LIMIT
 - DataReaders, 619
- DDS_LOST_BY_REMOTE_WRITERS_PER_SAMPLE_LIMIT
 - DataReaders, 620
- DDS_LOST_BY_SAMPLES_LIMIT
 - DataReaders, 622
- DDS_LOST_BY_SAMPLES_PER_INSTANCE_LIMIT
 - DataReaders, 621
- DDS_LOST_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT
 - DataReaders, 620
- DDS_LOST_BY_UNKNOWN_INSTANCE
 - DataReaders, 621
- DDS_LOST_BY_VIRTUAL_WRITERS_LIMIT
 - DataReaders, 620
- DDS_LOST_BY_WRITER
 - DataReaders, 618
- DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS
 - LIVELINESS, 1297
- DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS
 - LIVELINESS, 1297
- DDS_MEMORY_PERSISTENT_JOURNAL
 - DURABILITY, 1286
- DDS_MONITORING_LIBRARY_COMMAND_SERVICE_REQUEST
 - ServiceRequest Built-in Topic, 1101
- DDS_MONITORING_LIBRARY_REPLY_SERVICE_REQUEST
 - ServiceRequest Built-in Topic, 1101
- DDS_MONITORING_QOS_POLICY_ID
 - All QoS Policies, 1249
- DDS_MONITORING_QOS_POLICY_NAME
 - MONITORING, 1300
- DDS_MonitoringDedicatedParticipantSettings, 1836
 - collector_initial_peers, 1837
 - domain_id, 1837
 - enable, 1836
 - participant_qos_profile_name, 1837
- DDS_MonitoringDistributionSettings, 1838
 - dedicated_participant, 1839
 - event_settings, 1839
 - logging_settings, 1840
 - periodic_settings, 1840
 - publisher_qos_profile_name, 1839
- DDS_MonitoringEventDistributionSettings, 1840
 - concurrency_level, 1841
 - datawriter_qos_profile_name, 1842
 - publication_period, 1843
 - thread, 1842
- DDS_MonitoringLoggingDistributionSettings, 1843
 - concurrency_level, 1844
 - datawriter_qos_profile_name, 1845
 - max_historical_logs, 1845
 - publication_period, 1846
 - thread, 1846
- DDS_MonitoringLoggingForwardingSettings, 1847
 - middleware_forwarding_level, 1847
 - security_event_forwarding_level, 1847
 - service_forwarding_level, 1848
 - user_forwarding_level, 1848
- DDS_MonitoringMetricSelection, 1849
 - disabled_metrics_selection, 1850
 - enabled_metrics_selection, 1850
 - resource_selection, 1849
- DDS_MonitoringMetricSelectionSeq, 1851
- DDS_MonitoringPeriodicDistributionSettings, 1852
 - datawriter_qos_profile_name, 1852
 - polling_period, 1853
 - thread, 1852
- DDS_MonitoringQosPolicy, 1853
 - application_name, 1854
 - distribution_settings, 1854
 - enable, 1854
 - telemetry_data, 1855
- DDS_MonitoringTelemetryData, 1855
 - logs, 1856
 - metrics, 1855
- DDS_MULTICHANNEL_QOS_POLICY_ID
 - All QoS Policies, 1248
- DDS_MULTICHANNEL_QOS_POLICY_NAME
 - MULTICHANNEL, 1300
- DDS_MultiChannelQosPolicy, 1856
 - channels, 1858
 - filter_name, 1858
- DDS_MultiTopic
 - Topics, 185

- DDS_MultiTopic_as_topicdescription
 - Topics, 206
- DDS_MultiTopic_get_expression_parameters
 - Topics, 207
- DDS_MultiTopic_get_subscription_expression
 - Topics, 207
- DDS_MultiTopic_narrow
 - Topics, 207
- DDS_MultiTopic_set_expression_parameters
 - Topics, 208
- DDS_MUTABLE_EXTENSIBILITY
 - Type Code Support, 244
- DDS_NEW_VIEW_STATE
 - View States, 713
- DDS_NO_EXCEPTION_CODE
 - Exception Codes, 1219
- DDS_NO_INSTANCE_REMOVAL
 - DATA_READER_RESOURCE_LIMITS, 1255
- DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE
 - Exception Codes, 1220
- DDS_NO_RECOVER_INSTANCE_STATE_CONSISTENCY
 - RELIABILITY, 1324
- DDS_NO_REMOTE_PARTICIPANT_PURGE
 - DISCOVERY_CONFIG, 1283
- DDS_NO_REPLACEMENT_IGNORED_ENTITY_REPLACEMENT
 - DOMAIN_PARTICIPANT_RESOURCE_LIMITS, 1284
- DDS_NO_SERVICE_QOS
 - SERVICE, 1327
- DDS_NORMAL_PERSISTENT_SYNCHRONIZATION
 - DURABILITY, 1287
- DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE
 - Instance States, 715
- DDS_NOT_ALIVE_FIRST_IGNORED_ENTITY_REPLACEMENT
 - DOMAIN_PARTICIPANT_RESOURCE_LIMITS, 1284
- DDS_NOT_ALIVE_INSTANCE_STATE
 - Instance States, 715
- DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE
 - Instance States, 715
- DDS_NOT_LOST
 - DataReaders, 618
- DDS_NOT_NEW_VIEW_STATE
 - View States, 713
- DDS_NOT_READ_SAMPLE_STATE
 - Sample States, 711
- DDS_NOT_REJECTED
 - DataReaders, 622
- DDS_NOT_SET_CDR_PADDING
 - TYPESUPPORT, 1342
- DDS_OBSERVABILITY_COLLECTOR_SERVICE_QOS
 - SERVICE, 1327
- DDS_Octet
 - DDS-Specific Primitive Types, 1202
- DDS_OctetBuffer_alloc
 - Octet Buffer Support, 1502
- DDS_OctetBuffer_dup
 - Octet Buffer Support, 1502
- DDS_OctetBuffer_free
 - Octet Buffer Support, 1503
- DDS_Octets, 1859
 - Octets Built-in Type, 1154
- DDS_Octets_delete
 - Octets Built-in Type, 1156
- DDS_Octets_new
 - Octets Built-in Type, 1155
- DDS_Octets_new_w_size
 - Octets Built-in Type, 1155
- DDS_OctetsDataReader
 - Octets Built-in Type, 1154
- DDS_OctetsDataReader_as_datareader
 - Octets Built-in Type, 1166
- DDS_OctetsDataReader_narrow
 - Octets Built-in Type, 1166
- DDS_OctetsDataReader_read
 - Octets Built-in Type, 1166
- DDS_OctetsDataReader_read_next_sample
 - Octets Built-in Type, 1167
- DDS_OctetsDataReader_read_w_condition
 - Octets Built-in Type, 1167
- DDS_OctetsDataReader_return_loan
 - Octets Built-in Type, 1168
- DDS_OctetsDataReader_take
 - Octets Built-in Type, 1166
- DDS_OctetsDataReader_take_next_sample
 - Octets Built-in Type, 1168
- DDS_OctetsDataReader_take_w_condition
 - Octets Built-in Type, 1167
- DDS_OctetsDataWriter
 - Octets Built-in Type, 1154
- DDS_OctetsDataWriter_as_datawriter
 - Octets Built-in Type, 1160
- DDS_OctetsDataWriter_create_data
 - Octets Built-in Type, 1161
- DDS_OctetsDataWriter_delete_data
 - Octets Built-in Type, 1161
- DDS_OctetsDataWriter_narrow
 - Octets Built-in Type, 1160
- DDS_OctetsDataWriter_write
 - Octets Built-in Type, 1161
- DDS_OctetsDataWriter_write_octets
 - Octets Built-in Type, 1162
- DDS_OctetsDataWriter_write_octets_seq
 - Octets Built-in Type, 1162
- DDS_OctetsDataWriter_write_octets_seq_w_params
 - Octets Built-in Type, 1165
- DDS_OctetsDataWriter_write_octets_seq_w_timestamp
 - Octets Built-in Type, 1164

- DDS_OctetsDataWriter_write_octets_w_params
 - Octets Built-in Type, 1164
- DDS_OctetsDataWriter_write_octets_w_timestamp
 - Octets Built-in Type, 1163
- DDS_OctetsDataWriter_write_w_params
 - Octets Built-in Type, 1164
- DDS_OctetsDataWriter_write_w_timestamp
 - Octets Built-in Type, 1163
- DDS_OctetSeq, 1860
- DDS_OctetsSeq, 1860
- DDS_OctetsTypeSupport, 1860
- DDS_OctetsTypeSupport_data_to_string
 - Octets Built-in Type, 1160
- DDS_OctetsTypeSupport_deserialize_data_from_cdr_buffer
 - Octets Built-in Type, 1159
- DDS_OctetsTypeSupport_get_type_name
 - Octets Built-in Type, 1158
- DDS_OctetsTypeSupport_get_typecode
 - Octets Built-in Type, 1158
- DDS_OctetsTypeSupport_print_data
 - Octets Built-in Type, 1158
- DDS_OctetsTypeSupport_register_type
 - Octets Built-in Type, 1156
- DDS_OctetsTypeSupport_serialize_data_to_cdr_buffer
 - Octets Built-in Type, 1159
- DDS_OctetsTypeSupport_serialize_data_to_cdr_buffer_ex
 - Octets Built-in Type, 1159
- DDS_OctetsTypeSupport_unregister_type
 - Octets Built-in Type, 1157
- DDS_OFF_PERSISTENT_JOURNAL
 - DURABILITY, 1286
- DDS_OFF_PERSISTENT_SYNCHRONIZATION
 - DURABILITY, 1287
- DDS_OFFERED_DEADLINE_MISSED_STATUS
 - Status Kinds, 1228
- DDS_OFFERED_INCOMPATIBLE_QOS_STATUS
 - Status Kinds, 1229
- DDS_OfferedDeadlineMissedStatus, 1861
 - last_instance_handle, 1862
 - total_count, 1861
 - total_count_change, 1861
- DDS_OfferedDeadlineMissedStatus_copy
 - Data Writers, 511
- DDS_OfferedDeadlineMissedStatus_equals
 - Data Writers, 512
- DDS_OfferedDeadlineMissedStatus_finalize
 - Data Writers, 512
- DDS_OfferedDeadlineMissedStatus_initialize
 - Data Writers, 510
- DDS_OfferedDeadlineMissedStatus_INITIALIZER
 - Data Writers, 479
- DDS_OfferedIncompatibleQosStatus, 1862
 - last_policy_id, 1863
 - policies, 1863
 - total_count, 1863
 - total_count_change, 1863
- DDS_OfferedIncompatibleQosStatus_copy
 - Data Writers, 516
- DDS_OfferedIncompatibleQosStatus_equals
 - Data Writers, 517
- DDS_OfferedIncompatibleQosStatus_finalize
 - Data Writers, 516
- DDS_OfferedIncompatibleQosStatus_initialize
 - Data Writers, 515
- DDS_OfferedIncompatibleQosStatus_INITIALIZER
 - Data Writers, 480
- DDS_ON_DEMAND_FLOW_CONTROLLER_NAME
 - Flow Controllers, 565
- DDS_OWNERSHIP_QOS_POLICY_ID
 - All QoS Policies, 1246
- DDS_OWNERSHIP_QOS_POLICY_NAME
 - OWNERSHIP, 1302
- DDS_OwnershipQosPolicy, 1863
 - kind, 1868
- DDS_OwnershipQosPolicyKind
 - OWNERSHIP, 1301
- DDS_OWNERSHIPSTRENGTH_QOS_POLICY_ID
 - All QoS Policies, 1247
- DDS_OWNERSHIPSTRENGTH_QOS_POLICY_NAME
 - OWNERSHIP_STRENGTH, 1303
- DDS_OwnershipStrengthQosPolicy, 1868
 - value, 1869
- DDS_PARTICIPANT_CONFIG_PARAMS_DEFAULT
 - DomainParticipantFactory, 65
- DDS_PARTICIPANT_QOS_DEFAULT
 - DomainParticipantFactory, 65
- DDS_PARTICIPANT_TOPIC_NAME
 - Participant Built-in Topics, 1092
- DDS_ParticipantBuiltinTopicData, 1870
 - available_built_in_endpoints_ext, 1872
 - dds_built_in_endpoints, 1872
 - default_unicast_locators, 1873
 - domain_id, 1873
 - key, 1871
 - partial_configuration, 1875
 - Participant Built-in Topics, 1091
 - participant_name, 1873
 - partition, 1874
 - product_version, 1873
 - property, 1871
 - reachability_lease_duration, 1874
 - rtps_protocol_version, 1871
 - rtps_vendor_id, 1871
 - transport_info, 1873
 - trust_algorithm_info, 1874
 - trust_protection_info, 1874
 - user_data, 1871
- DDS_ParticipantBuiltinTopicDataDataReader

- Participant Built-in Topics, 1092
- DDS_ParticipantBuiltinTopicDataSeq, 1876
- DDS_ParticipantBuiltinTopicDataSupport, 1876
- DDS_ParticipantTrustAlgorithmInfo, 1876
 - Built-in Topic's Trust Types, 312
 - interceptor, 1877
 - key_establishment, 1877
 - signature, 1877
- DDS_ParticipantTrustAttributesMask
 - Built-in Topic's Trust Types, 310
- DDS_ParticipantTrustInterceptorAlgorithmInfo, 1877
 - Built-in Topic's Trust Types, 312
 - builtin_endpoints_required_mask, 1878
 - builtin_kx_endpoints_required_mask, 1878
 - supported_mask, 1878
 - user_endpoints_default_required_mask, 1878
- DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo, 1879
 - Built-in Topic's Trust Types, 312
 - shared_secret, 1879
- DDS_ParticipantTrustProtectionInfo, 1879
 - bitmask, 1880
 - Built-in Topic's Trust Types, 310
 - plugin_bitmask, 1880
- DDS_ParticipantTrustSignatureAlgorithmInfo, 1880
 - Built-in Topic's Trust Types, 311
 - message_auth, 1881
 - trust_chain, 1881
- DDS_PARTITION_QOS_POLICY_ID
 - All QoS Policies, 1247
- DDS_PARTITION_QOS_POLICY_NAME
 - PARTITION, 1303
- DDS_PartitionQosPolicy, 1881
 - name, 1883
- DDS_PERSIST_PERSISTENT_JOURNAL
 - DURABILITY, 1286
- DDS_PERSISTENCE_SERVICE_QOS
 - SERVICE, 1327
- DDS_PERSISTENT_DURABILITY_QOS
 - DURABILITY, 1288
- DDS_PersistentJournalKind
 - DURABILITY, 1286
- DDS_PersistentStorageSettings, 1883
 - enable, 1885
 - file_name, 1885
 - journal_kind, 1886
 - reader_checkpoint_frequency, 1889
 - restore, 1887
 - synchronization_kind, 1886
 - trace_file_name, 1885
 - vacuum, 1886
 - writer_instance_cache_allocation, 1887
 - writer_memory_state, 1889
 - writer_sample_cache_allocation, 1888
- DDS_PersistentSynchronizationKind
 - DURABILITY, 1286
- DDS_PluginEndpointTrustAttributesMask
 - Built-in Topic's Trust Types, 311
- DDS_PluginParticipantTrustAttributesMask
 - Built-in Topic's Trust Types, 310
- DDS_PRESENTATION_QOS_POLICY_ID
 - All QoS Policies, 1246
- DDS_PRESENTATION_QOS_POLICY_NAME
 - PRESENTATION, 1305
- DDS_PresentationQosPolicy, 1890
 - access_scope, 1893
 - coherent_access, 1893
 - drop_incomplete_coherent_set, 1894
 - ordered_access, 1893
- DDS_PresentationQosPolicyAccessScopeKind
 - PRESENTATION, 1304
- DDS_PRINT_FORMAT_PROPERTY_DEFAULT
 - Topics, 209
- DDS_PrintFormatKind
 - Topics, 186
- DDS_PrintFormatProperty, 1894
 - enum_as_int, 1896
 - include_root_elements, 1896
 - kind, 1895
 - pretty_print, 1895
 - Topics, 175
- DDS_PrintFormatProperty_INITIALIZER
 - Topics, 172
- DDS_PRIVATE_MEMBER
 - Type Code Support, 239
- DDS_ProductVersion_t, 1897
 - major, 1897
 - minor, 1897
 - release, 1897
 - revision, 1898
- DDS_PRODUCTVERSION_UNKNOWN
 - Common types and functions, 1106
- DDS_PROFILE_BASELINE
 - Builtin Qos Profiles, 1400
- DDS_PROFILE_BASELINE_5_0_0
 - Builtin Qos Profiles, 1400
- DDS_PROFILE_BASELINE_5_1_0
 - Builtin Qos Profiles, 1400
- DDS_PROFILE_BASELINE_5_2_0
 - Builtin Qos Profiles, 1401
- DDS_PROFILE_BASELINE_5_3_0
 - Builtin Qos Profiles, 1401
- DDS_PROFILE_BASELINE_6_0_0
 - Builtin Qos Profiles, 1401
- DDS_PROFILE_BASELINE_6_1_0
 - Builtin Qos Profiles, 1401
- DDS_PROFILE_BASELINE_7_0_0
 - Builtin Qos Profiles, 1401

DDS_PROFILE_BASELINE_7_1_0	DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA
Builtin Qos Profiles, 1402	Builtin Qos Profiles, 1406
DDS_PROFILE_BASELINE_ROOT	DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_FAST_FLOW
Builtin Qos Profiles, 1400	Builtin Qos Profiles, 1407
DDS_PROFILE_GENERIC_510_TRANSPORT_COMPATIBILITY	DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_MEDIUM_FLOW
Builtin Qos Profiles, 1403	Builtin Qos Profiles, 1407
DDS_PROFILE_GENERIC_AUTO_TUNING	DDS_PROFILE_GENERIC_STRICT_RELIABLE_LARGE_DATA_SLOW_FLOW
Builtin Qos Profiles, 1409	Builtin Qos Profiles, 1408
DDS_PROFILE_GENERIC_BEST_EFFORT	DDS_PROFILE_GENERIC_STRICT_RELIABLE_LOW_LATENCY
Builtin Qos Profiles, 1404	Builtin Qos Profiles, 1405
DDS_PROFILE_GENERIC_COMMON	DDS_PROFILE_PATTERN_ALARM_EVENT
Builtin Qos Profiles, 1402	Builtin Qos Profiles, 1413
DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY	DDS_PROFILE_PATTERN_ALARM_STATUS
Builtin Qos Profiles, 1402	Builtin Qos Profiles, 1414
DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY	DDS_PROFILE_PATTERN_EVENT
Builtin Qos Profiles, 1403	Builtin Qos Profiles, 1413
DDS_PROFILE_GENERIC_CONNEXT_MICRO_COMPATIBILITY	DDS_PROFILE_PATTERN_LAST_VALUE_CACHE
Builtin Qos Profiles, 1403	Builtin Qos Profiles, 1414
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE	DDS_PROFILE_PATTERN_PERIODIC_DATA
Builtin Qos Profiles, 1404	Builtin Qos Profiles, 1412
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA	DDS_PROFILE_PATTERN_RELIABLE_STREAMING
Builtin Qos Profiles, 1407	Builtin Qos Profiles, 1413
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA	DDS_PROFILE_PATTERN_RPC
Builtin Qos Profiles, 1408	Builtin Qos Profiles, 1412
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA	DDS_PROFILE_PATTERN_STATUS
Builtin Qos Profiles, 1408	Builtin Qos Profiles, 1414
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_LARGE_DATA	DDS_PROFILE_PATTERN_STREAMING
Builtin Qos Profiles, 1408	Builtin Qos Profiles, 1412
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_PERSONALITY	DDS_PROFILE_QOS_POLICY_ID
Builtin Qos Profiles, 1409	All QoS Policies, 1248
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT	DDS_PROFILE_QOS_POLICY_NAME
Builtin Qos Profiles, 1409	PROFILE, 1306
DDS_PROFILE_GENERIC_KEEP_LAST_RELIABLE_TRANSIENT	DDS_ProfileQosPolicy, 1898
Builtin Qos Profiles, 1409	ignore_environment_profile, 1899
DDS_PROFILE_GENERIC_MINIMAL_MEMORY_FOOTPRINT	ignore_resource_profile, 1900
Builtin Qos Profiles, 1410	ignore_user_profile, 1899
DDS_PROFILE_GENERIC_MONITORING2	string_profile, 1899
Builtin Qos Profiles, 1410	url_profile, 1899
DDS_PROFILE_GENERIC_MONITORING2_WAN	DDS_PROPERTY_QOS_IMMUTABLE
Builtin Qos Profiles, 1411	PROPERTY, 1310
DDS_PROFILE_GENERIC_MONITORING_COMMON	DDS_PROPERTY_QOS_MUTABLE
Builtin Qos Profiles, 1402	PROPERTY, 1310
DDS_PROFILE_GENERIC_OTHER_DDS_VENDOR_COMPATIBILITY	DDS_PROPERTY_QOS_MUTABLE_UNTIL_ENABLE
Builtin Qos Profiles, 1403	PROPERTY, 1310
DDS_PROFILE_GENERIC_PARTICIPANT_LARGE_DATA	DDS_PROPERTY_QOS_POLICY_ID
Builtin Qos Profiles, 1405	All QoS Policies, 1248
DDS_PROFILE_GENERIC_PARTICIPANT_LARGE_DATA_MEDIUM_DATA	DDS_PROPERTY_QOS_POLICY_NAME
Builtin Qos Profiles, 1406	PROPERTY, 1317
DDS_PROFILE_GENERIC_SECURITY	DDS_Property_t, 1900
Builtin Qos Profiles, 1410	name, 1901
DDS_PROFILE_GENERIC_STRICT_RELIABLE	propagate, 1901
Builtin Qos Profiles, 1404	value, 1901
DDS_PROFILE_GENERIC_STRICT_RELIABLE_HIGH_THROUGHPUT	DDS_PropertyQosPolicy, 1901
Builtin Qos Profiles, 1405	PROPERTY, 1308

- value, 1903
- DDS_PropertyQosPolicyHelper_add_pointer_property
PROPERTY, 1312
- DDS_PropertyQosPolicyHelper_add_property
PROPERTY, 1311
- DDS_PropertyQosPolicyHelper_assert_pointer_property
PROPERTY, 1312
- DDS_PropertyQosPolicyHelper_assert_property
PROPERTY, 1310
- DDS_PropertyQosPolicyHelper_get_number_of_properties
PROPERTY, 1310
- DDS_PropertyQosPolicyHelper_get_properties
PROPERTY, 1315
- DDS_PropertyQosPolicyHelper_get_properties_into_policy
PROPERTY, 1315
- DDS_PropertyQosPolicyHelper_get_property_mutability
PROPERTY, 1316
- DDS_PropertyQosPolicyHelper_lookup_property
PROPERTY, 1313
- DDS_PropertyQosPolicyHelper_lookup_property_with_prefix
PROPERTY, 1314
- DDS_PropertyQosPolicyHelper_remove_property
PROPERTY, 1314
- DDS_PropertyQosPolicyMutability
PROPERTY, 1309
- DDS_PropertySeq, 1903
- DDS_PROTOCOL_ACKNOWLEDGMENT_MODE
RELIABILITY, 1323
- DDS_PROTOCOLVERSION
 - Common types and functions, 1105
- DDS_PROTOCOLVERSION_1_0
 - Common types and functions, 1104
- DDS_PROTOCOLVERSION_1_1
 - Common types and functions, 1105
- DDS_PROTOCOLVERSION_1_2
 - Common types and functions, 1105
- DDS_PROTOCOLVERSION_2_0
 - Common types and functions, 1105
- DDS_PROTOCOLVERSION_2_1
 - Common types and functions, 1105
- DDS_ProtocolVersion_t, 1904
 - Common types and functions, 1106
 - major, 1904
 - minor, 1904
- DDS_PUBLIC_MEMBER
 - Type Code Support, 240
- DDS_PUBLICATION_MATCHED_STATUS
 - Status Kinds, 1232
- DDS_PUBLICATION_PRIORITY_AUTOMATIC
PUBLISH_MODE, 1318
- DDS_PUBLICATION_PRIORITY_UNDEFINED
PUBLISH_MODE, 1318
- DDS_PUBLICATION_TOPIC_NAME
 - Publication Built-in Topics, 1096
- DDS_PublicationBuiltinTopicData, 1905
 - data_tags, 1910
 - deadline, 1908
 - destination_order, 1909
 - disable_positive_acks, 1912
 - durability, 1907
 - durability_service, 1908
 - group_data, 1910
 - key, 1907
 - latency_budget, 1908
 - lifespan, 1908
 - liveliness, 1908
 - locator_filter, 1912
 - ownership, 1909
 - ownership_strength, 1909
 - participant_key, 1907
 - partition, 1910
 - presentation, 1909
 - product_version, 1912
 - property, 1911
 - Publication Built-in Topics, 1095
 - publication_name, 1912
 - publisher_key, 1911
 - related_datareader_key, 1913
 - reliability, 1908
 - representation, 1910
 - rtps_protocol_version, 1912
 - rtps_vendor_id, 1912
 - service, 1911
 - topic_data, 1910
 - topic_name, 1907
 - trust_algorithm_info, 1913
 - trust_protection_info, 1913
 - type_code, 1910
 - type_name, 1907
 - unicast_locators, 1911
 - user_data, 1909
 - virtual_guid, 1911
- DDS_PublicationBuiltinTopicDataReader
 - Publication Built-in Topics, 1095
- DDS_PublicationBuiltinTopicDataSeq, 1914
- DDS_PublicationBuiltinTopicDataTypeSupport, 1914
- DDS_PublicationMatchedStatus, 1914
 - current_count, 1916
 - current_count_change, 1916
 - current_count_peak, 1916
 - last_subscription_handle, 1916
 - total_count, 1915
 - total_count_change, 1916
- DDS_PublicationMatchedStatus_copy
 - Data Writers, 518
- DDS_PublicationMatchedStatus_equals
 - Data Writers, 519
- DDS_PublicationMatchedStatus_finalize

- Data Writers, 519
- DDS_PublicationMatchedStatus_initialize
 - Data Writers, 518
- DDS_PublicationMatchedStatus_INITIALIZER
 - Data Writers, 481
- DDS_Publisher
 - Publishers, 443
- DDS_Publisher_as_entity
 - Publishers, 448
- DDS_Publisher_begin_coherent_changes
 - Publishers, 459
- DDS_Publisher_copy_from_topic_qos
 - Publishers, 462
- DDS_Publisher_create_datawriter
 - Publishers, 453
- DDS_Publisher_create_datawriter_with_profile
 - Publishers, 455
- DDS_Publisher_delete_contained_entities
 - Publishers, 461
- DDS_Publisher_delete_datawriter
 - Publishers, 456
- DDS_Publisher_end_coherent_changes
 - Publishers, 460
- DDS_Publisher_get_all_datawriters
 - Publishers, 460
- DDS_Publisher_get_default_datawriter_qos
 - Publishers, 449
- DDS_Publisher_get_default_library
 - Publishers, 464
- DDS_Publisher_get_default_profile
 - Publishers, 465
- DDS_Publisher_get_default_profile_library
 - Publishers, 465
- DDS_Publisher_get_listener
 - Publishers, 466
- DDS_Publisher_get_listenerX
 - Publishers, 467
- DDS_Publisher_get_participant
 - Publishers, 461
- DDS_Publisher_get_qos
 - Publishers, 464
- DDS_Publisher_lookup_datawriter
 - Publishers, 457
- DDS_Publisher_lookup_datawriter_by_name
 - Publishers, 469
- DDS_PUBLISHER_QOS_DEFAULT
 - DomainParticipants, 161
- DDS_PUBLISHER_QOS_PRINT_ALL
 - DomainParticipants, 163
- DDS_Publisher_resume_publications
 - Publishers, 458
- DDS_Publisher_set_default_datawriter_qos
 - Publishers, 450
- DDS_Publisher_set_default_datawriter_qos_with_profile
 - Publishers, 450
- DDS_Publisher_set_default_library
 - Publishers, 452
- DDS_Publisher_set_default_profile
 - Publishers, 451
- DDS_Publisher_set_listener
 - Publishers, 466
- DDS_Publisher_set_qos
 - Publishers, 462
- DDS_Publisher_set_qos_with_profile
 - Publishers, 463
- DDS_Publisher_suspend_publications
 - Publishers, 457
- DDS_Publisher_wait_for_acknowledgments
 - Publishers, 467
- DDS_Publisher_wait_for_asynchronous_publishing
 - Publishers, 468
- DDS_PublisherListener, 1917
 - as_datawriterlistener, 1918
- DDS_PublisherListener_INITIALIZER
 - Publishers, 442
- DDS_PublisherQos, 1918
 - asynchronous_publisher, 1919
 - entity_factory, 1919
 - group_data, 1919
 - partition, 1919
 - presentation, 1919
 - publisher_name, 1919
- DDS_PublisherQos_copy
 - Publishers, 448
- DDS_PublisherQos_equals
 - Publishers, 444
- DDS_PublisherQos_finalize
 - Publishers, 447
- DDS_PublisherQos_initialize
 - Publishers, 446
- DDS_PublisherQos_INITIALIZER
 - Publishers, 442
- DDS_PublisherQos_print
 - Publishers, 444
- DDS_PublisherQos_to_string
 - Publishers, 445
- DDS_PublisherQos_to_string_w_params
 - Publishers, 446
- DDS_PublisherSeq, 1920
- DDS_PUBLISHMODE_QOS_POLICY_ID
 - All QoS Policies, 1248
- DDS_PUBLISHMODE_QOS_POLICY_NAME
 - PUBLISH_MODE, 1319
- DDS_PublishModeQosPolicy, 1920
 - flow_controller_name, 1922
 - kind, 1922
 - priority, 1923
- DDS_PublishModeQosPolicyKind

- PUBLISH_MODE, 1318
- DDS_QOS_ELEMENT_NAME_USE_XML_CONFIG
 - DomainParticipantConfigParams, 1431
- DDS_QOS_POLICY_COUNT
 - All QoS Policies, 1246
- DDS_QosPolicyCount, 1924
 - count, 1925
 - policy_id, 1925
- DDS_QosPolicyCountSeq, 1925
- DDS_QosPolicyId_t
 - All QoS Policies, 1246
- DDS_QosPrintFormat, 1926
 - indent, 1927
 - is_standalone, 1926
 - print_private, 1927
- DDS_QosPrintFormat_INITIALIZER
 - All QoS Policies, 1245
- DDS_QueryCondition
 - Query Conditions, 699
- DDS_QueryCondition_as_readcondition
 - Query Conditions, 699
- DDS_QueryCondition_get_query_expression
 - Query Conditions, 699
- DDS_QueryCondition_get_query_parameters
 - Query Conditions, 700
- DDS_QueryCondition_set_query_parameters
 - Query Conditions, 700
- DDS_QueryConditionParams, 1928
 - as_readconditionparams, 1928
 - query_expression, 1929
 - query_parameters, 1929
- DDS_QUEUEING_SERVICE_QOS
 - SERVICE, 1327
- DDS_READ_SAMPLE_STATE
 - Sample States, 711
- DDS_ReadCondition
 - Read Conditions, 694
- DDS_ReadCondition_as_condition
 - Read Conditions, 695
- DDS_ReadCondition_get_datareader
 - Read Conditions, 696
- DDS_ReadCondition_get_instance_state_mask
 - Read Conditions, 696
- DDS_ReadCondition_get_sample_state_mask
 - Read Conditions, 695
- DDS_ReadCondition_get_stream_kind_mask
 - Read Conditions, 696
- DDS_ReadCondition_get_view_state_mask
 - Read Conditions, 695
- DDS_ReadConditionParams, 1929
 - instance_states, 1930
 - sample_states, 1930
 - stream_kinds, 1930
 - view_states, 1930

- DDS_READERDATALIFECYCLE_QOS_POLICY_ID
 - All QoS Policies, 1247
- DDS_READERDATALIFECYCLE_QOS_POLICY_NAME
 - READER_DATA_LIFECYCLE, 1320
- DDS_ReaderDataLifecycleQosPolicy, 1930
 - autopurge_disposed_instances_delay, 1932
 - autopurge_disposed_samples_delay, 1932
 - autopurge_nowriter_instances_delay, 1933
 - autopurge_nowriter_samples_delay, 1932
- DDS_RECEIVERPOOL_QOS_POLICY_ID
 - All QoS Policies, 1248
- DDS_RECEIVERPOOL_QOS_POLICY_NAME
 - RECEIVER_POOL, 1321
- DDS_ReceiverPoolQosPolicy, 1934
 - buffer_alignment, 1936
 - buffer_size, 1935
 - thread, 1935
- DDS_RECORDING_SERVICE_QOS
 - SERVICE, 1327
- DDS_RECOVER_INSTANCE_STATE_CONSISTENCY
 - RELIABILITY, 1324
- DDS_REDELIVERED_SAMPLE
 - Sample Flags, 1388
- DDS_REJECTED_BY_DECODE_FAILURE
 - DataReaders, 624
- DDS_REJECTED_BY_INSTANCES_LIMIT
 - DataReaders, 622
- DDS_REJECTED_BY_REMOTE_WRITER_SAMPLES_PER_VIRTUAL_QOS
 - DataReaders, 623
- DDS_REJECTED_BY_SAMPLES_LIMIT
 - DataReaders, 622
- DDS_REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT
 - DataReaders, 623
- DDS_REJECTED_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT
 - DataReaders, 623
- DDS_RELIABILITY_QOS_POLICY_ID
 - All QoS Policies, 1247
- DDS_RELIABILITY_QOS_POLICY_NAME
 - RELIABILITY, 1325
- DDS_ReliabilityQosPolicy, 1936
 - acknowledgment_kind, 1939
 - instance_state_consistency_kind, 1940
 - kind, 1939
 - max_blocking_time, 1939
- DDS_ReliabilityQosPolicyAcknowledgmentModeKind
 - RELIABILITY, 1323
- DDS_ReliabilityQosPolicyKind
 - RELIABILITY, 1322
- DDS_RELIABLE_READER_ACTIVITY_CHANGED_STATUS
 - Status Kinds, 1234
- DDS_RELIABLE_RELIABILITY_QOS
 - RELIABILITY, 1323
- DDS_RELIABLE_WRITER_CACHE_CHANGED_STATUS
 - Status Kinds, 1234

- DDS_ReliableReaderActivityChangedStatus, 1940
 - active_count, 1941
 - active_count_change, 1942
 - inactive_count, 1941
 - inactive_count_change, 1942
 - last_instance_handle, 1942
- DDS_ReliableReaderActivityChangedStatus_copy
 - Data Writers, 526
- DDS_ReliableReaderActivityChangedStatus_equals
 - Data Writers, 528
- DDS_ReliableReaderActivityChangedStatus_finalize
 - Data Writers, 526
- DDS_ReliableReaderActivityChangedStatus_initialize
 - Data Writers, 525
- DDS_ReliableReaderActivityChangedStatus_INITIALIZER
 - Data Writers, 482
- DDS_ReliableWriterCacheChangedStatus, 1942
 - empty_reliable_writer_cache, 1943
 - full_reliable_writer_cache, 1944
 - high_watermark_reliable_writer_cache, 1944
 - low_watermark_reliable_writer_cache, 1944
 - replaced_unacknowledged_sample_count, 1945
 - unacknowledged_sample_count, 1944
 - unacknowledged_sample_count_peak, 1945
- DDS_ReliableWriterCacheChangedStatus_copy
 - Data Writers, 523
- DDS_ReliableWriterCacheChangedStatus_equals
 - Data Writers, 525
- DDS_ReliableWriterCacheChangedStatus_finalize
 - Data Writers, 524
- DDS_ReliableWriterCacheChangedStatus_initialize
 - Data Writers, 523
- DDS_ReliableWriterCacheChangedStatus_INITIALIZER
 - Data Writers, 482
- DDS_ReliableWriterCacheEventCount, 1945
 - total_count, 1946
 - total_count_change, 1946
- DDS_ReliableWriterCacheEventCount_equals
 - Data Writers, 522
- DDS_RemoteParticipantPurgeKind
 - DISCOVERY_CONFIG, 1282
- DDS_REPLAY_SERVICE_QOS
 - SERVICE, 1327
- DDS_REPLICATE_SAMPLE
 - Sample Flags, 1388
- DDS_REQUESTED_DEADLINE_MISSED_STATUS
 - Status Kinds, 1229
- DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS
 - Status Kinds, 1229
- DDS_RequestedDeadlineMissedStatus, 1946
 - last_instance_handle, 1947
 - total_count, 1946
 - total_count_change, 1947
- DDS_RequestedDeadlineMissedStatus_copy
 - Data Readers, 650
- DDS_RequestedDeadlineMissedStatus_equals
 - Data Readers, 651
- DDS_RequestedDeadlineMissedStatus_finalize
 - Data Readers, 651
- DDS_RequestedDeadlineMissedStatus_initialize
 - Data Readers, 650
- DDS_RequestedDeadlineMissedStatus_INITIALIZER
 - Data Readers, 610
- DDS_RequestedIncompatibleQosStatus, 1947
 - last_policy_id, 1948
 - policies, 1949
 - total_count, 1948
 - total_count_change, 1948
- DDS_RequestedIncompatibleQosStatus_copy
 - Data Readers, 654
- DDS_RequestedIncompatibleQosStatus_equals
 - Data Readers, 655
- DDS_RequestedIncompatibleQosStatus_finalize
 - Data Readers, 654
- DDS_RequestedIncompatibleQosStatus_initialize
 - Data Readers, 653
- DDS_RequestedIncompatibleQosStatus_INITIALIZER
 - Data Readers, 611
- DDS_RESOURCELIMITS_QOS_POLICY_ID
 - All QoS Policies, 1247
- DDS_RESOURCELIMITS_QOS_POLICY_NAME
 - RESOURCE_LIMITS, 1325
- DDS_ResourceLimitsQosPolicy, 1949
 - initial_instances, 1952
 - initial_samples, 1952
 - instance_hash_buckets, 1953
 - max_instances, 1951
 - max_samples, 1951
 - max_samples_per_instance, 1952
- DDS_RETCODE_ALREADY_DELETED
 - Return Codes, 1222
- DDS_RETCODE_BAD_PARAMETER
 - Return Codes, 1222
- DDS_RETCODE_ERROR
 - Return Codes, 1222
- DDS_RETCODE_ILLEGAL_OPERATION
 - Return Codes, 1222
- DDS_RETCODE_IMMUTABLE_POLICY
 - Return Codes, 1222
- DDS_RETCODE_INCONSISTENT_POLICY
 - Return Codes, 1222
- DDS_RETCODE_NO_DATA
 - Return Codes, 1222
- DDS_RETCODE_NOT_ALLOWED_BY_SECURITY
 - Return Codes, 1222
- DDS_RETCODE_NOT_ENABLED
 - Return Codes, 1222
- DDS_RETCODE_OK

- Return Codes, 1222
- DDS_RETCODE_OUT_OF_RESOURCES
 - Return Codes, 1222
- DDS_RETCODE_PRECONDITION_NOT_MET
 - Return Codes, 1222
- DDS_RETCODE_TIMEOUT
 - Return Codes, 1222
- DDS_RETCODE_UNSUPPORTED
 - Return Codes, 1222
- DDS_ReturnCode_t
 - Return Codes, 1221
- DDS_ROUTING_SERVICE_QOS
 - SERVICE, 1327
- DDS_RR_FLOW_CONTROLLER_SCHED_POLICY
 - Flow Controllers, 560
- DDS_RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS
 - WIRE_PROTOCOL, 1348
- DDS_RTPS_AUTO_ID_FROM_IP
 - WIRE_PROTOCOL, 1348
- DDS_RTPS_AUTO_ID_FROM_MAC
 - WIRE_PROTOCOL, 1348
- DDS_RTPS_AUTO_ID_FROM_UUID
 - WIRE_PROTOCOL, 1348
- DDS_RTPS_EntityId_t, 1953
 - GUID Support, 1211
- DDS_RTPS_GUID_t, 1954
 - GUID Support, 1211
- DDS_RTPS_GuidPrefix_t
 - GUID Support, 1211
- DDS_RTPS_RESERVED_PORT_BUILTIN_MULTICAST
 - WIRE_PROTOCOL, 1348
- DDS_RTPS_RESERVED_PORT_BUILTIN_UNICAST
 - WIRE_PROTOCOL, 1348
- DDS_RTPS_RESERVED_PORT_MASK_ALL
 - WIRE_PROTOCOL, 1346
- DDS_RTPS_RESERVED_PORT_MASK_DEFAULT
 - WIRE_PROTOCOL, 1345
- DDS_RTPS_RESERVED_PORT_MASK_NONE
 - WIRE_PROTOCOL, 1346
- DDS_RTPS_RESERVED_PORT_USER_MULTICAST
 - WIRE_PROTOCOL, 1348
- DDS_RTPS_RESERVED_PORT_USER_UNICAST
 - WIRE_PROTOCOL, 1348
- DDS_RtpsReliableReaderProtocol_t, 1954
 - app_ack_period, 1958
 - heartbeat_suppression_duration, 1956
 - max_heartbeat_response_delay, 1955
 - min_app_ack_response_keep_duration, 1958
 - min_heartbeat_response_delay, 1955
 - nack_period, 1956
 - receive_window_size, 1957
 - round_trip_time, 1957
 - samples_per_app_ack, 1959
- DDS_RtpsReliableWriterProtocol_t, 1960
 - disable_positive_acks_decrease_sample_keep_duration_factor, 1972
 - disable_positive_acks_enable_adaptive_sample_keep_duration, 1971
 - disable_positive_acks_increase_sample_keep_duration_factor, 1972
 - disable_positive_acks_max_sample_keep_duration, 1971
 - disable_positive_acks_min_sample_keep_duration, 1970
 - disable_repair_piggyback_heartbeat, 1978
 - enable_multicast_periodic_heartbeat, 1977
 - fast_heartbeat_period, 1963
 - heartbeat_period, 1963
 - heartbeats_per_max_samples, 1967
 - high_watermark, 1962
 - inactivate_nonprogressing_readers, 1966
 - late_joiner_heartbeat_period, 1964
 - low_watermark, 1961
 - max_bytes_per_nack_response, 1969
 - max_heartbeat_retries, 1966
 - max_nack_response_delay, 1968
 - max_send_window_size, 1973
 - min_nack_response_delay, 1968
 - min_send_window_size, 1973
 - multicast_resend_threshold, 1977
 - nack_suppression_duration, 1969
 - samples_per_virtual_heartbeat, 1965
 - send_window_decrease_factor, 1976
 - send_window_increase_factor, 1975
 - send_window_update_period, 1974
 - virtual_heartbeat_period, 1965
- DDS_RtpsReservedPortKind
 - WIRE_PROTOCOL, 1347
- DDS_RtpsReservedPortKindMask
 - WIRE_PROTOCOL, 1347
- DDS_RtpsWellKnownPorts_t, 1978
 - builtin_multicast_port_offset, 1982
 - builtin_unicast_port_offset, 1982
 - domain_id_gain, 1981
 - participant_id_gain, 1982
 - port_base, 1980
 - user_multicast_port_offset, 1983
 - user_unicast_port_offset, 1983
- DDS_SAMPLE_LOST_STATUS
 - Status Kinds, 1230
- DDS_SAMPLE_REJECTED_STATUS
 - Status Kinds, 1230
- DDS_SampleFlag
 - Sample Flags, 1387
- DDS_SampleFlagBits
 - Sample Flags, 1387, 1388
- DDS_SampleHandler, 1983
 - handler_data, 1984

- on_new_sample, 1984
- DDS_SampleHandler_INITIALIZER
 - SampleProcessor, 1504
- DDS_SampleIdentity_equals
 - WriteParams, 1389
- DDS_SampleIdentity_t, 1984
- DDS_SampleInfo, 1985
 - absolute_generation_rank, 1991
 - coherent_set_info, 1995
 - disposed_generation_count, 1990
 - flag, 1994
 - generation_rank, 1991
 - instance_handle, 1990
 - instance_state, 1989
 - no_writers_generation_count, 1990
 - original_publication_virtual_guid, 1993
 - original_publication_virtual_sequence_number, 1993
 - publication_handle, 1990
 - publication_sequence_number, 1992
 - reception_sequence_number, 1993
 - reception_timestamp, 1992
 - related_original_publication_virtual_guid, 1993
 - related_original_publication_virtual_sequence_number, 1994
 - related_source_guid, 1994
 - related_subscription_guid, 1995
 - sample_rank, 1991
 - sample_state, 1988
 - source_guid, 1994
 - source_timestamp, 1989
 - topic_query_guid, 1995
 - valid_data, 1992
 - view_state, 1989
- DDS_SampleInfo_get_related_sample_identity
 - Data Samples, 702
- DDS_SampleInfo_get_sample_identity
 - Data Samples, 702
- DDS_SampleInfoSeq, 1996
- DDS_SampleLostStatus, 1996
 - last_reason, 1997
 - total_count, 1996
 - total_count_change, 1997
- DDS_SampleLostStatus_copy
 - DataReaders, 656
- DDS_SampleLostStatus_equals
 - DataReaders, 657
- DDS_SampleLostStatus_finalize
 - DataReaders, 656
- DDS_SampleLostStatus_initialize
 - DataReaders, 655
- DDS_SampleLostStatus_INITIALIZER
 - DataReaders, 612
- DDS_SampleLostStatusKind
 - DataReaders, 618
- DDS_SampleProcessor
 - SampleProcessor, 1505
- DDS_SampleProcessor_attach_reader
 - SampleProcessor, 1506
- DDS_SampleProcessor_delete
 - SampleProcessor, 1509
- DDS_SampleProcessor_detach_reader
 - SampleProcessor, 1507
- DDS_SampleProcessor_get_datareaders
 - SampleProcessor, 1508
- DDS_SampleProcessor_lookup_sample_handler
 - SampleProcessor, 1508
- DDS_SampleProcessor_new
 - SampleProcessor, 1510
- DDS_SampleProcessor_new_with_aws
 - SampleProcessor, 1510
- DDS_SampleRejectedStatus, 1997
 - last_instance_handle, 1998
 - last_reason, 1998
 - total_count, 1998
 - total_count_change, 1998
- DDS_SampleRejectedStatus_copy
 - DataReaders, 658
- DDS_SampleRejectedStatus_equals
 - DataReaders, 659
- DDS_SampleRejectedStatus_finalize
 - DataReaders, 658
- DDS_SampleRejectedStatus_initialize
 - DataReaders, 657
- DDS_SampleRejectedStatus_INITIALIZER
 - DataReaders, 612
- DDS_SampleRejectedStatusKind
 - DataReaders, 622
- DDS_SampleStateKind
 - Sample States, 710
- DDS_SampleStateMask
 - Sample States, 710
- DDS_SEQUENCE_INITIALIZER
 - Sequence Support, 1513
- DDS_SEQUENCE_NUMBER_MAX
 - Sequence Number Support, 1218
- DDS_SEQUENCE_NUMBER_UNKNOWN
 - Sequence Number Support, 1218
- DDS_SEQUENCE_NUMBER_ZERO
 - Sequence Number Support, 1218
- DDS_SequenceNumber_add
 - Sequence Number Support, 1215
- DDS_SequenceNumber_compare
 - Sequence Number Support, 1216
- DDS_SequenceNumber_minusminus
 - Sequence Number Support, 1216
- DDS_SequenceNumber_plusplus
 - Sequence Number Support, 1216
- DDS_SequenceNumber_subtract

- Sequence Number Support, 1215
- DDS_SequenceNumber_t, 1999
 - high, 1999
 - low, 1999
- Sequence Number Support, 1215
- DDS_SERVICE_QOS_POLICY_NAME
 - SERVICE, 1328
- DDS_SERVICE_REQUEST_ACCEPTED_STATUS
 - Status Kinds, 1233
- DDS_SERVICE_REQUEST_TOPIC_NAME
 - ServiceRequest Built-in Topic, 1101
- DDS_ServiceQosPolicy, 2000
 - kind, 2000
- DDS_ServiceQosPolicyKind
 - SERVICE, 1327
- DDS_ServiceRequest, 2001
 - instance_id, 2001
 - request_body, 2002
 - service_id, 2001
 - ServiceRequest Built-in Topic, 1100
- DDS_ServiceRequestAcceptedStatus, 2002
 - current_count, 2003
 - current_count_change, 2003
 - last_request_handle, 2003
 - service_id, 2004
 - total_count, 2003
 - total_count_change, 2003
- DDS_ServiceRequestAcceptedStatus_copy
 - Data Writers, 521
- DDS_ServiceRequestAcceptedStatus_equals
 - Data Writers, 522
- DDS_ServiceRequestAcceptedStatus_finalize
 - Data Writers, 521
- DDS_ServiceRequestAcceptedStatus_initialize
 - Data Writers, 520
- DDS_ServiceRequestAcceptedStatus_INITIALIZER
 - Data Writers, 481
- DDS_ServiceRequestDataReader
 - ServiceRequest Built-in Topic, 1100
- DDS_ServiceRequestSeq, 2004
- DDS_ServiceRequestTypeSupport, 2004
- DDS_SHARED_OWNERSHIP_QOS
 - OWNERSHIP, 1302
- DDS_Short
 - DDS-Specific Primitive Types, 1202
- DDS_ShortSeq, 2005
- DDS_SNIPPET_5_1_0_TRANSPORT_ENABLE
 - Builtin Qos Profiles, 1430
- DDS_SNIPPET_COMPATIBILITY_CONNEXT_MICRO_VERSION
 - Builtin Qos Profiles, 1429
- DDS_SNIPPET_COMPATIBILITY_OTHER_DDS_VENDORS
 - Builtin Qos Profiles, 1429
- DDS_SNIPPET_FEATURE_AUTO_TUNING_ENABLE
 - Builtin Qos Profiles, 1425

- DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_209MBPS
 - Builtin Qos Profiles, 1424
- DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_52MBPS
 - Builtin Qos Profiles, 1424
- DDS_SNIPPET_FEATURE_FLOW_CONTROLLER_838MBPS
 - Builtin Qos Profiles, 1424
- DDS_SNIPPET_FEATURE_MONITORING2_ENABLE
 - Builtin Qos Profiles, 1426
- DDS_SNIPPET_FEATURE_MONITORING_ENABLE
 - Builtin Qos Profiles, 1425
- DDS_SNIPPET_FEATURE_SECURITY_ENABLE
 - Builtin Qos Profiles, 1426
- DDS_SNIPPET_FEATURE_TOPIC_QUERY_ENABLE
 - Builtin Qos Profiles, 1426
- DDS_SNIPPET_OPTIMIZATION_DATACACHE_LARGE_DATA_DYNAMIC
 - Builtin Qos Profiles, 1418
- DDS_SNIPPET_OPTIMIZATION_DISCOVERY_COMMON
 - Builtin Qos Profiles, 1418
- DDS_SNIPPET_OPTIMIZATION_DISCOVERY_ENDPOINT_FAST
 - Builtin Qos Profiles, 1419
- DDS_SNIPPET_OPTIMIZATION_DISCOVERY_PARTICIPANT_COMPACT
 - Builtin Qos Profiles, 1419
- DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_COMMON
 - Builtin Qos Profiles, 1415
- DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_HIGH_RATE
 - Builtin Qos Profiles, 1416
- DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_ALL
 - Builtin Qos Profiles, 1415
- DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_KEEP_LAST
 - Builtin Qos Profiles, 1416
- DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LARGE_DATA
 - Builtin Qos Profiles, 1417
- DDS_SNIPPET_OPTIMIZATION_RELIABILITY_PROTOCOL_LOW_LATENCY
 - Builtin Qos Profiles, 1417
- DDS_SNIPPET_OPTIMIZATION_TRANSPORT_LARGE_BUFFERS
 - Builtin Qos Profiles, 1420
- DDS_SNIPPET_QOS_POLICY_BATCHING_ENABLE
 - Builtin Qos Profiles, 1423
- DDS_SNIPPET_QOS_POLICY_DURABILITY_PERSISTENT
 - Builtin Qos Profiles, 1423
- DDS_SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT
 - Builtin Qos Profiles, 1422
- DDS_SNIPPET_QOS_POLICY_DURABILITY_TRANSIENT_LOCAL
 - Builtin Qos Profiles, 1422
- DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_ALL
 - Builtin Qos Profiles, 1421
- DDS_SNIPPET_QOS_POLICY_HISTORY_KEEP_LAST_1
 - Builtin Qos Profiles, 1421
- DDS_SNIPPET_QOS_POLICY_PUBLISH_MODE_ASYNCHRONOUS
 - Builtin Qos Profiles, 1421
- DDS_SNIPPET_QOS_POLICY_RELIABILITY_BEST_EFFORT
 - Builtin Qos Profiles, 1420
- DDS_SNIPPET_QOS_POLICY_RELIABILITY_RELIABLE
 - Builtin Qos Profiles, 1420

- DDS_SNIPPET_TRANSPORT_TCP_LAN_CLIENT
 - Builtin Qos Profiles, 1427
- DDS_SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_QOS
 - Builtin Qos Profiles, 1428
- DDS_SNIPPET_TRANSPORT_TCP_WAN_ASYMMETRIC_QOS
 - Builtin Qos Profiles, 1428
- DDS_SNIPPET_TRANSPORT_TCP_WAN_SYMMETRIC_QOS
 - Builtin Qos Profiles, 1427
- DDS_SNIPPET_TRANSPORT_UDP_AVOID_IP_FRAGMENTATION
 - Builtin Qos Profiles, 1428
- DDS_SNIPPET_TRANSPORT_UDP_WAN
 - Builtin Qos Profiles, 1429
- DDS_SQLFILTER_NAME
 - DomainParticipants, 163
- DDS_STATUS_MASK_ALL
 - Status Kinds, 1226
- DDS_STATUS_MASK_NONE
 - Status Kinds, 1226
- DDS_StatusCondition
 - Conditions and WaitSets, 1371
- DDS_StatusCondition_as_condition
 - Conditions and WaitSets, 1377
- DDS_StatusCondition_get_enabled_statuses
 - Conditions and WaitSets, 1379
- DDS_StatusCondition_get_entity
 - Conditions and WaitSets, 1380
- DDS_StatusCondition_set_enabled_statuses
 - Conditions and WaitSets, 1379
- DDS_StatusKind
 - Status Kinds, 1227
- DDS_StatusMask
 - Status Kinds, 1226
- DDS_StreamKind
 - Stream Kinds, 716
- DDS_StreamKindMask
 - Stream Kinds, 716
- DDS_String_alloc
 - String Support, 1528
- DDS_String_dup
 - String Support, 1528
- DDS_String_free
 - String Support, 1529
- DDS_String_replace
 - String Support, 1529
- DDS_StringDataReader
 - String Built-in Type, 1114
- DDS_StringDataReader_as_datareader
 - String Built-in Type, 1121
- DDS_StringDataReader_narrow
 - String Built-in Type, 1120
- DDS_StringDataReader_read
 - String Built-in Type, 1121
- DDS_StringDataReader_read_next_sample
 - String Built-in Type, 1122
- DDS_StringDataReader_read_w_condition
 - String Built-in Type, 1122
- DDS_StringDataReader_return_loan
 - String Built-in Type, 1123
- DDS_StringDataReader_take
 - String Built-in Type, 1121
- DDS_StringDataReader_take_next_sample
 - String Built-in Type, 1123
- DDS_StringDataReader_take_w_condition
 - String Built-in Type, 1122
- DDS_StringDataWriter
 - String Built-in Type, 1114
- DDS_StringDataWriter_as_datawriter
 - String Built-in Type, 1118
- DDS_StringDataWriter_create_data
 - String Built-in Type, 1119
- DDS_StringDataWriter_delete_data
 - String Built-in Type, 1119
- DDS_StringDataWriter_narrow
 - String Built-in Type, 1118
- DDS_StringDataWriter_write
 - String Built-in Type, 1119
- DDS_StringDataWriter_write_w_params
 - String Built-in Type, 1120
- DDS_StringDataWriter_write_w_timestamp
 - String Built-in Type, 1120
- DDS_STRINGMATCHFILTER_NAME
 - DomainParticipants, 164
- DDS_StringSeq, 2005
- DDS_StringTypeSupport, 2006
- DDS_StringTypeSupport_data_to_string
 - String Built-in Type, 1118
- DDS_StringTypeSupport_deserialize_data_from_cdr_buffer
 - String Built-in Type, 1117
- DDS_StringTypeSupport_get_type_name
 - String Built-in Type, 1116
- DDS_StringTypeSupport_get_typecode
 - String Built-in Type, 1117
- DDS_StringTypeSupport_print_data
 - String Built-in Type, 1116
- DDS_StringTypeSupport_register_type
 - String Built-in Type, 1114
- DDS_StringTypeSupport_serialize_data_to_cdr_buffer
 - String Built-in Type, 1117
- DDS_StringTypeSupport_serialize_data_to_cdr_buffer_ex
 - String Built-in Type, 1117
- DDS_StringTypeSupport_unregister_type
 - String Built-in Type, 1115
- DDS_StructMember, 2007
 - bits, 2008
 - id, 2008
 - is_key, 2008
 - is_optional, 2008
 - is_pointer, 2008

- name, 2007
- type, 2007
- DDS_StructMemberSeq, 2009
- DDS_Subscriber
 - Subscribers, 572
- DDS_Subscriber_as_entity
 - Subscribers, 578
- DDS_Subscriber_begin_access
 - Subscribers, 587
- DDS_Subscriber_copy_from_topic_qos
 - Subscribers, 591
- DDS_Subscriber_create_datareader
 - Subscribers, 581
- DDS_Subscriber_create_datareader_with_profile
 - Subscribers, 583
- DDS_Subscriber_delete_contained_entities
 - Subscribers, 585
- DDS_Subscriber_delete_datareader
 - Subscribers, 585
- DDS_Subscriber_end_access
 - Subscribers, 588
- DDS_Subscriber_get_all_datareaders
 - Subscribers, 589
- DDS_Subscriber_get_datareaders
 - Subscribers, 588
- DDS_Subscriber_get_default_datareader_qos
 - Subscribers, 579
- DDS_Subscriber_get_default_library
 - Subscribers, 596
- DDS_Subscriber_get_default_profile
 - Subscribers, 594
- DDS_Subscriber_get_default_profile_library
 - Subscribers, 595
- DDS_Subscriber_get_listener
 - Subscribers, 597
- DDS_Subscriber_get_listenerX
 - Subscribers, 598
- DDS_Subscriber_get_participant
 - Subscribers, 590
- DDS_Subscriber_get_qos
 - Subscribers, 593
- DDS_Subscriber_lookup_datareader
 - Subscribers, 586
- DDS_Subscriber_lookup_datareader_by_name
 - Subscribers, 598
- DDS_Subscriber_notify_datareaders
 - Subscribers, 590
- DDS_SUBSCRIBER_QOS_DEFAULT
 - DomainParticipants, 162
- DDS_SUBSCRIBER_QOS_PRINT_ALL
 - DomainParticipants, 164
- DDS_Subscriber_set_default_datareader_qos
 - Subscribers, 580
- DDS_Subscriber_set_default_datareader_qos_with_profile
 - Subscribers, 580
- DDS_Subscriber_set_default_library
 - Subscribers, 595
- DDS_Subscriber_set_default_profile
 - Subscribers, 594
- DDS_Subscriber_set_listener
 - Subscribers, 597
- DDS_Subscriber_set_qos
 - Subscribers, 592
- DDS_Subscriber_set_qos_with_profile
 - Subscribers, 592
- DDS_SubscriberListener, 2009
 - as_datareaderlistener, 2010
 - on_data_on_readers, 2010
- DDS_SubscriberListener_DataOnReadersCallback
 - Subscribers, 573
- DDS_SubscriberListener_INITIALIZER
 - Subscribers, 572
- DDS_SubscriberQos, 2010
 - entity_factory, 2011
 - group_data, 2011
 - partition, 2011
 - presentation, 2011
 - subscriber_name, 2011
- DDS_SubscriberQos_copy
 - Subscribers, 578
- DDS_SubscriberQos_equals
 - Subscribers, 573
- DDS_SubscriberQos_finalize
 - Subscribers, 577
- DDS_SubscriberQos_initialize
 - Subscribers, 577
- DDS_SubscriberQos_INITIALIZER
 - Subscribers, 571
- DDS_SubscriberQos_print
 - Subscribers, 575
- DDS_SubscriberQos_to_string
 - Subscribers, 575
- DDS_SubscriberQos_to_string_w_params
 - Subscribers, 576
- DDS_SubscriberSeq, 2012
- DDS_SUBSCRIPTION_MATCHED_STATUS
 - Status Kinds, 1232
- DDS_SUBSCRIPTION_TOPIC_NAME
 - Subscription Built-in Topics, 1098
- DDS_SubscriptionBuiltinTopicData, 2012
 - content_filter_property, 2019
 - data_tags, 2017
 - deadline, 2015
 - destination_order, 2016
 - disable_positive_acks, 2020
 - durability, 2015
 - group_data, 2017
 - key, 2014

- latency_budget, 2015
- liveliness, 2015
- multicast_locators, 2018
- ownership, 2016
- participant_key, 2014
- partition, 2017
- presentation, 2016
- product_version, 2020
- property, 2018
- related_datawriter_key, 2021
- reliability, 2015
- representation, 2017
- rtps_protocol_version, 2019
- rtps_vendor_id, 2019
- service, 2019
- subscriber_key, 2018
- Subscription Built-in Topics, 1097
- subscription_name, 2020
- time_based_filter, 2016
- topic_data, 2017
- topic_name, 2014
- trust_algorithm_info, 2020
- trust_protection_info, 2020
- type_code, 2018
- type_consistency, 2017
- type_name, 2014
- unicast_locators, 2018
- user_data, 2016
- virtual_guid, 2019
- DDS_SubscriptionBuiltinTopicDataDataReader
 - Subscription Built-in Topics, 1097
- DDS_SubscriptionBuiltinTopicDataSeq, 2021
- DDS_SubscriptionBuiltinTopicDataTypeSupport, 2021
- DDS_SubscriptionMatchedStatus, 2022
 - current_count, 2023
 - current_count_change, 2024
 - current_count_peak, 2023
 - last_publication_handle, 2024
 - total_count, 2023
 - total_count_change, 2023
- DDS_SubscriptionMatchedStatus_copy
 - DataReaders, 660
- DDS_SubscriptionMatchedStatus_equals
 - DataReaders, 660
- DDS_SubscriptionMatchedStatus_finalize
 - DataReaders, 660
- DDS_SubscriptionMatchedStatus_initialize
 - DataReaders, 659
- DDS_SubscriptionMatchedStatus_INITIALIZER
 - DataReaders, 613
- DDS_SYNCHRONOUS_PUBLISH_MODE_QOS
 - PUBLISH_MODE, 1319
- DDS_SYSTEM_EXCEPTION_CODE
 - Exception Codes, 1219
- DDS_SYSTEMRESOURCELIMITS_QOS_POLICY_ID
 - All QoS Policies, 1248
- DDS_SYSTEMRESOURCELIMITS_QOS_POLICY_NAME
 - SYSTEM_RESOURCE_LIMITS, 1328
- DDS_SystemResourceLimitsQosPolicy, 2024
 - initial_objects_per_thread, 2025
 - max_objects_per_thread, 2025
- DDS_Tag, 2026
 - name, 2026
 - value, 2027
- DDS_TagSeq, 2027
- DDS_TCKind
 - Type Code Support, 243
- DDS_TheParticipantFactory
 - DomainParticipantFactory, 32
- DDS_THREAD_SETTINGS_CANCEL_ASYNCHRONOUS
 - Thread Settings, 1237
- DDS_THREAD_SETTINGS_CPU_NO_ROTATION
 - Thread Settings, 1238
- DDS_THREAD_SETTINGS_CPU_RR_ROTATION
 - Thread Settings, 1238
- DDS_THREAD_SETTINGS_FLOATING_POINT
 - Thread Settings, 1237
- DDS_THREAD_SETTINGS_KIND_MASK_DEFAULT
 - Thread Settings, 1236
- DDS_THREAD_SETTINGS_PRIORITY_ENFORCE
 - Thread Settings, 1237
- DDS_THREAD_SETTINGS_REALTIME_PRIORITY
 - Thread Settings, 1237
- DDS_THREAD_SETTINGS_STUDIO
 - Thread Settings, 1237
- DDS_ThreadFactory, 2027
 - create_thread, 2028
 - delete_thread, 2028
 - factory_data, 2028
- DDS_ThreadFactory_CreateThreadCallback
 - User-managed Threads, 1433
- DDS_ThreadFactory_DeleteThreadCallback
 - User-managed Threads, 1434
- DDS_ThreadFactory_INITIALIZER
 - User-managed Threads, 1433
- DDS_ThreadFactory_OnSpawnedFunction
 - User-managed Threads, 1433
- DDS_ThreadSettings_t, 2029
 - cpu_list, 2030
 - cpu_rotation, 2030
 - mask, 2029
 - priority, 2030
 - stack_size, 2030
- DDS_ThreadSettingsCpuRotationKind
 - Thread Settings, 1237
- DDS_ThreadSettingsKind
 - Thread Settings, 1237
- DDS_ThreadSettingsKindMask

- Thread Settings, 1236
- DDS_TIME_INVALID
 - Time Support, 1208
- DDS_TIME_INVALID_NSEC
 - Time Support, 1208
- DDS_TIME_INVALID_SEC
 - Time Support, 1208
- DDS_Time_is_invalid
 - Time Support, 1206
- DDS_Time_is_zero
 - Time Support, 1206
- DDS_TIME_MAX
 - Time Support, 1207
- DDS_Time_t, 2031
 - nanosec, 2031
 - sec, 2031
- DDS_TIME_ZERO
 - Time Support, 1206
- DDS_TIMEBASEDFILTER_QOS_POLICY_ID
 - All QoS Policies, 1247
- DDS_TIMEBASEDFILTER_QOS_POLICY_NAME
 - TIME_BASED_FILTER, 1329
- DDS_TimeBasedFilterQosPolicy, 2032
 - minimum_separation, 2034
- DDS_TK_ALIAS
 - Type Code Support, 243
- DDS_TK_ARRAY
 - Type Code Support, 243
- DDS_TK_BOOLEAN
 - Type Code Support, 243
- DDS_TK_CHAR
 - Type Code Support, 243
- DDS_TK_DOUBLE
 - Type Code Support, 243
- DDS_TK_ENUM
 - Type Code Support, 243
- DDS_TK_FLOAT
 - Type Code Support, 243
- DDS_TK_INT8
 - Type Code Support, 243
- DDS_TK_LONG
 - Type Code Support, 243
- DDS_TK_LONGDOUBLE
 - Type Code Support, 243
- DDS_TK_LONGLONG
 - Type Code Support, 243
- DDS_TK_NULL
 - Type Code Support, 243
- DDS_TK_OCTET
 - Type Code Support, 243
- DDS_TK_SEQUENCE
 - Type Code Support, 243
- DDS_TK_SHORT
 - Type Code Support, 243
- DDS_TK_STRING
 - Type Code Support, 243
- DDS_TK_STRUCT
 - Type Code Support, 243
- DDS_TK_UINT8
 - Type Code Support, 243
- DDS_TK_ULONG
 - Type Code Support, 243
- DDS_TK_ULONGLONG
 - Type Code Support, 243
- DDS_TK_UNION
 - Type Code Support, 243
- DDS_TK_USHORT
 - Type Code Support, 243
- DDS_TK_VALUE
 - Type Code Support, 243
- DDS_TK_WCHAR
 - Type Code Support, 243
- DDS_TK_WSTRING
 - Type Code Support, 243
- DDS_Topic
 - Topics, 175
- DDS_Topic_as_entity
 - Topics, 195
- DDS_Topic_as_topicdescription
 - Topics, 195
- DDS_Topic_get_inconsistent_topic_status
 - Topics, 196
- DDS_Topic_get_listener
 - Topics, 200
- DDS_Topic_get_listenerX
 - Topics, 200
- DDS_Topic_get_qos
 - Topics, 199
- DDS_Topic_narrow
 - Topics, 196
- DDS_Topic_narrow_from_entity
 - Topics, 196
- DDS_TOPIC_PRESENTATION_QOS
 - PRESENTATION, 1305
- DDS_TOPIC_QOS_DEFAULT
 - DomainParticipants, 160
- DDS_TOPIC_QOS_PRINT_ALL
 - DomainParticipants, 164
- DDS_TOPIC_QUERY_SELECTION_KIND_CONTINUOUS
 - Topic Queries, 708
- DDS_TOPIC_QUERY_SELECTION_KIND_HISTORY_SNAPSHOT
 - Topic Queries, 708
- DDS_TOPIC_QUERY_SELECTION_SELECT_ALL
 - Topic Queries, 709
- DDS_TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER
 - Topic Queries, 709
- DDS_TOPIC_QUERY_SERVICE_REQUEST_ID
 - ServiceRequest Built-in Topic, 1101

- DDS_TOPIC_QUERY_STREAM
 - Stream Kinds, 716
- DDS_TOPIC_SCOPE_DESTINATIONORDER_QOS
 - DESTINATION_ORDER, 1275
- DDS_Topic_set_listener
 - Topics, 199
- DDS_Topic_set_qos
 - Topics, 197
- DDS_Topic_set_qos_with_profile
 - Topics, 198
- DDS_TOPIC_TOPIC_NAME
 - Topic Built-in Topics, 1094
- DDS_TopicBuiltinTopicData, 2034
 - deadline, 2037
 - destination_order, 2038
 - durability, 2036
 - durability_service, 2037
 - history, 2038
 - key, 2036
 - latency_budget, 2037
 - lifespan, 2038
 - liveliness, 2037
 - name, 2036
 - ownership, 2038
 - reliability, 2037
 - representation, 2039
 - resource_limits, 2038
 - Topic Built-in Topics, 1093
 - topic_data, 2038
 - transport_priority, 2037
 - type_name, 2036
- DDS_TopicBuiltinTopicDataDataReader
 - Topic Built-in Topics, 1093
- DDS_TopicBuiltinTopicDataSeq, 2039
- DDS_TopicBuiltinTopicDataTypeSupport, 2039
- DDS_TOPICDATA_QOS_POLICY_ID
 - All QoS Policies, 1247
- DDS_TOPICDATA_QOS_POLICY_NAME
 - TOPIC_DATA, 1330
- DDS_TopicDataQosPolicy, 2040
 - value, 2041
- DDS_TopicDescription
 - Topics, 175
- DDS_TopicDescription_get_name
 - Topics, 194
- DDS_TopicDescription_get_participant
 - Topics, 195
- DDS_TopicDescription_get_type_name
 - Topics, 193
- DDS_TopicListener, 2041
 - as_listener, 2042
 - on_inconsistent_topic, 2042
- DDS_TopicListener_InconsistentTopicCallback
 - Topics, 176
- DDS_TopicListener_INITIALIZER
 - Topics, 174
- DDS_TopicQos, 2042
 - deadline, 2044
 - destination_order, 2044
 - durability, 2043
 - durability_service, 2044
 - history, 2045
 - latency_budget, 2044
 - lifespan, 2045
 - liveliness, 2044
 - ownership, 2045
 - reliability, 2044
 - representation, 2045
 - resource_limits, 2045
 - topic_data, 2043
 - transport_priority, 2045
- DDS_TopicQos_copy
 - Topics, 193
- DDS_TopicQos_equals
 - Topics, 189
- DDS_TopicQos_finalize
 - Topics, 192
- DDS_TopicQos_initialize
 - Topics, 191
- DDS_TopicQos_INITIALIZER
 - Topics, 173
- DDS_TopicQos_print
 - Topics, 189
- DDS_TopicQos_to_string
 - Topics, 190
- DDS_TopicQos_to_string_w_params
 - Topics, 190
- DDS_TopicQuery
 - Topic Queries, 707
- DDS_TopicQuery_get_guid
 - Topic Queries, 708
- DDS_TopicQueryData, 2046
 - original_related_reader_guid, 2047
 - Topic Queries, 707
 - topic_name, 2046
 - topic_query_selection, 2046
- DDS_TOPICQUERYDISPATCH_QOS_POLICY_ID
 - All QoS Policies, 1249
- DDS_TOPICQUERYDISPATCH_QOS_POLICY_NAME
 - TOPIC_QUERY_DISPATCH, 1331
- DDS_TopicQueryDispatchQosPolicy, 2047
 - enable, 2048
 - publication_period, 2048
 - samples_per_period, 2049
- DDS_TopicQueryHelper_topic_query_data_from_service_request
 - Topic Queries, 708
- DDS_TopicQuerySelection, 2049
 - filter_class_name, 2050

- filter_expression, 2050
- filter_parameters, 2050
- kind, 2051
- Topic Queries, 706
- DDS_TopicQuerySelectionKind
 - Topic Queries, 707
- DDS_TRANSIENT_DURABILITY_QOS
 - DURABILITY, 1288
- DDS_TRANSIENT_LOCAL_DURABILITY_QOS
 - DURABILITY, 1288
- DDS_TRANSPORTBUILTIN_MASK_ALL
 - TRANSPORT_BUILTIN, 1333
- DDS_TRANSPORTBUILTIN_MASK_DEFAULT
 - TRANSPORT_BUILTIN, 1332
- DDS_TRANSPORTBUILTIN_MASK_NONE
 - TRANSPORT_BUILTIN, 1332
- DDS_TRANSPORTBUILTIN_QOS_POLICY_ID
 - All QoS Policies, 1248
- DDS_TRANSPORTBUILTIN_QOS_POLICY_NAME
 - TRANSPORT_BUILTIN, 1334
- DDS_TRANSPORTBUILTIN_SHMEM
 - TRANSPORT_BUILTIN, 1334
- DDS_TRANSPORTBUILTIN_SHMEM_ALIAS
 - TRANSPORT_BUILTIN, 1334
- DDS_TRANSPORTBUILTIN_UDPv4
 - TRANSPORT_BUILTIN, 1334
- DDS_TRANSPORTBUILTIN_UDPv4_ALIAS
 - TRANSPORT_BUILTIN, 1334
- DDS_TRANSPORTBUILTIN_UDPv4_WAN
 - TRANSPORT_BUILTIN, 1334
- DDS_TRANSPORTBUILTIN_UDPv4_WAN_ALIAS
 - TRANSPORT_BUILTIN, 1334
- DDS_TRANSPORTBUILTIN_UDPv6
 - TRANSPORT_BUILTIN, 1334
- DDS_TRANSPORTBUILTIN_UDPv6_ALIAS
 - TRANSPORT_BUILTIN, 1334
- DDS_TransportBuiltinKind
 - TRANSPORT_BUILTIN, 1333
- DDS_TransportBuiltinKindMask
 - TRANSPORT_BUILTIN, 1333
- DDS_TransportBuiltinQosPolicy, 2051
 - mask, 2052
- DDS_TransportInfo_t, 2052
 - class_id, 2052
 - message_size_max, 2053
- DDS_TransportInfoSeq, 2053
- DDS_TRANSPORTMULTICAST_QOS_POLICY_ID
 - All QoS Policies, 1248
- DDS_TRANSPORTMULTICAST_QOS_POLICY_NAME
 - TRANSPORT_MULTICAST, 1336
- DDS_TRANSPORTMULTICASTMAPPING_QOS_POLICY_ID
 - All QoS Policies, 1248
- DDS_TRANSPORTMULTICASTMAPPING_QOS_POLICY_NAME
 - TRANSPORT_MULTICAST_MAPPING, 1337
- DDS_TransportMulticastMapping_t, 2053
 - addresses, 2054
 - mapping_function, 2055
 - topic_expression, 2054
- DDS_TransportMulticastMappingFunction_t, 2055
 - dll, 2055
 - function_name, 2056
- DDS_TransportMulticastMappingQosPolicy, 2056
 - value, 2058
- DDS_TransportMulticastMappingSeq, 2058
- DDS_TransportMulticastQosPolicy, 2058
 - kind, 2059
 - value, 2059
- DDS_TransportMulticastQosPolicyKind
 - TRANSPORT_MULTICAST, 1336
- DDS_TransportMulticastSettings_t, 2060
 - receive_address, 2061
 - receive_port, 2061
 - transports, 2061
- DDS_TransportMulticastSettingsSeq, 2062
- DDS_TRANSPORTPRIORITY_QOS_POLICY_ID
 - All QoS Policies, 1247
- DDS_TRANSPORTPRIORITY_QOS_POLICY_NAME
 - TRANSPORT_PRIORITY, 1338
- DDS_TransportPriorityQosPolicy, 2062
 - value, 2064
- DDS_TRANSPORTSELECTION_QOS_POLICY_ID
 - All QoS Policies, 1248
- DDS_TRANSPORTSELECTION_QOS_POLICY_NAME
 - TRANSPORT_SELECTION, 1338
- DDS_TransportSelectionQosPolicy, 2064
 - enabled_transports, 2065
- DDS_TRANSPORTUNICAST_QOS_POLICY_ID
 - All QoS Policies, 1248
- DDS_TRANSPORTUNICAST_QOS_POLICY_NAME
 - TRANSPORT_UNICAST, 1339
- DDS_TransportUnicastQosPolicy, 2065
 - value, 2067
- DDS_TransportUnicastSettings_t, 2067
 - receive_port, 2068
 - transports, 2068
- DDS_TransportUnicastSettingsSeq, 2069
- DDS_TRUNCATE_PERSISTENT_JOURNAL
 - DURABILITY, 1286
- DDS_TrustAlgorithmBit
 - Built-in Topic's Trust Types, 311
- DDS_TrustAlgorithmRequirements, 2069
 - Built-in Topic's Trust Types, 311
 - required_mask, 2069
 - supported_mask, 2069
- DDS_TrustAlgorithmSet
 - Built-in Topic's Trust Types, 311
- DDS_TYPE_CODE_PRINT_KIND_IDL
 - Type Code Support, 243

- DDS_TypeCode_print_kind_XML
 - Type Code Support, 243
- DDS_TypeConsistencyEnforcementQoSPolicy
 - All QoS Policies, 1247
- DDS_TypeConsistencyEnforcementQoSPolicyTypeConsistencyEnforcement, 1341
- DDS_TypeAllocationParams_t, 2070
 - allocate_optional_members, 2070
 - allocate_pointers, 2070
- DDS_TypeCode, 2071
- DDS_TypeCode_add_member
 - Type Code Support, 280
- DDS_TypeCode_add_member_ex
 - Type Code Support, 282
- DDS_TypeCode_add_member_to_enum
 - Type Code Support, 278
- DDS_TypeCode_add_member_to_union
 - Type Code Support, 279
- DDS_TypeCode_array_dimension
 - Type Code Support, 268
- DDS_TypeCode_array_dimension_count
 - Type Code Support, 268
- DDS_TypeCode_cdr_serialized_sample_key_max_size
 - Type Code Support, 288
- DDS_TypeCode_cdr_serialized_sample_max_size
 - Type Code Support, 287
- DDS_TypeCode_cdr_serialized_sample_min_size
 - Type Code Support, 288
- DDS_TypeCode_concrete_base_type
 - Type Code Support, 272
- DDS_TypeCode_content_type
 - Type Code Support, 270
- DDS_TypeCode_default_annotation
 - Type Code Support, 249
- DDS_TypeCode_default_index
 - Type Code Support, 271
- DDS_TypeCode_default_value
 - Type Code Support, 250
- DDS_TypeCode_discriminator_type
 - Type Code Support, 266
- DDS_TypeCode_element_count
 - Type Code Support, 269
- DDS_TypeCode_equal
 - Type Code Support, 247
- DDS_TypeCode_extensibility_kind
 - Type Code Support, 245
- DDS_TypeCode_find_member_by_id
 - Type Code Support, 275
- DDS_TypeCode_find_member_by_name
 - Type Code Support, 256
- DDS_TypeCode_get_cdr_serialized_sample_max_size
 - Type Code Support, 286
- DDS_TypeCode_get_type_object_serialized_size
 - Type Code Support, 277
- DDS_TypeCode_index_invalid
 - Type Code Support, 238
- DDS_TypeCode_is_alias_pointer
 - Type Code Support, 270
- DDS_TypeCode_is_member_bitfield
 - Type Code Support, 263
- DDS_TypeCode_is_member_key
 - Type Code Support, 261
- DDS_TypeCode_is_member_pointer
 - Type Code Support, 262
- DDS_TypeCode_is_member_required
 - Type Code Support, 262
- DDS_TypeCode_key_member
 - Type Code Support, 240
- DDS_TypeCode_kind
 - Type Code Support, 244
- DDS_TypeCode_length
 - Type Code Support, 267
- DDS_TypeCode_max_annotation
 - Type Code Support, 250
- DDS_TypeCode_max_value
 - Type Code Support, 252
- DDS_TypeCode_member_bitfield_bits
 - Type Code Support, 264
- DDS_TypeCode_member_count
 - Type Code Support, 255
- DDS_TypeCode_member_default_annotation
 - Type Code Support, 249
- DDS_TypeCode_member_default_value
 - Type Code Support, 252
- DDS_TypeCode_member_id
 - Type Code Support, 275
- DDS_TypeCode_member_id_invalid
 - Type Code Support, 237
- DDS_TypeCode_member_label
 - Type Code Support, 259
- DDS_TypeCode_member_label_count
 - Type Code Support, 258
- DDS_TypeCode_member_max_value
 - Type Code Support, 254
- DDS_TypeCode_member_min_value
 - Type Code Support, 253
- DDS_TypeCode_member_name
 - Type Code Support, 256
- DDS_TypeCode_member_ordinal
 - Type Code Support, 260
- DDS_TypeCode_member_type
 - Type Code Support, 257
- DDS_TypeCode_member_visibility
 - Type Code Support, 265
- DDS_TypeCode_min_annotation
 - Type Code Support, 250
- DDS_TypeCode_min_value
 - Type Code Support, 251

- DDS_TypeCode_name
 - Type Code Support, 248
- DDS_TYPECODE_NONKEY_MEMBER
 - Type Code Support, 240
- DDS_TYPECODE_NONKEY_REQUIRED_MEMBER
 - Type Code Support, 241
- DDS_TYPECODE_NOT_BITFIELD
 - Type Code Support, 238
- DDS_TypeCode_print
 - Type Code Support, 284
- DDS_TypeCode_print_IDL
 - Type Code Support, 283
- DDS_TypeCode_PrintFormat_INITIALIZER
 - Type Code Support, 241
- DDS_TypeCode_to_string
 - Type Code Support, 285
- DDS_TypeCode_to_string_w_format
 - Type Code Support, 286
- DDS_TypeCode_type_modifier
 - Type Code Support, 274
- DDS_TypeCodeFactory, 2071
- DDS_TypeCodeFactory_clone_tc
 - Type Code Support, 289
- DDS_TypeCodeFactory_create_alias_tc
 - Type Code Support, 297
- DDS_TypeCodeFactory_create_array_tc
 - Type Code Support, 300
- DDS_TypeCodeFactory_create_enum_tc
 - Type Code Support, 296
- DDS_TypeCodeFactory_create_enum_tc_ex
 - Type Code Support, 296
- DDS_TypeCodeFactory_create_sequence_tc
 - Type Code Support, 299
- DDS_TypeCodeFactory_create_string_tc
 - Type Code Support, 298
- DDS_TypeCodeFactory_create_struct_tc
 - Type Code Support, 291
- DDS_TypeCodeFactory_create_struct_tc_ex
 - Type Code Support, 292
- DDS_TypeCodeFactory_create_union_tc
 - Type Code Support, 294
- DDS_TypeCodeFactory_create_union_tc_ex
 - Type Code Support, 295
- DDS_TypeCodeFactory_create_value_tc
 - Type Code Support, 293
- DDS_TypeCodeFactory_create_value_tc_ex
 - Type Code Support, 293
- DDS_TypeCodeFactory_create_wstring_tc
 - Type Code Support, 298
- DDS_TypeCodeFactory_delete_tc
 - Type Code Support, 290
- DDS_TypeCodeFactory_finalize_instance
 - Type Code Support, 289
- DDS_TypeCodeFactory_get_instance
 - Type Code Support, 289
- DDS_TypeCodeFactory_get_primitive_tc
 - Type Code Support, 291
- DDS_TypeCodePrintFormatKind
 - Type Code Support, 242
- DDS_TypeCodePrintFormatProperty, 2073
 - indent, 2073
 - print_complete_type, 2075
 - print_kind, 2074
 - print_ordinals, 2073
- DDS_TypeConsistencyEnforcementQosPolicy, 2076
 - force_type_validation, 2079
 - ignore_enum_literal_names, 2079
 - ignore_member_names, 2078
 - ignore_sequence_bounds, 2077
 - ignore_string_bounds, 2078
 - kind, 2077
 - prevent_type_widening, 2078
- DDS_TypeConsistencyKind
 - TYPE_CONSISTENCY_ENFORCEMENT, 1340
- DDS_TypeDeallocationParams_t, 2080
 - delete_optional_members, 2080
 - delete_pointers, 2080
- DDS_TypeSupport
 - User Data Type Support, 214
- DDS_TYPESUPPORT_C
 - User Data Type Support, 213
- DDS_TYPESUPPORT_QOS_POLICY_ID
 - All QoS Policies, 1248
- DDS_TYPESUPPORT_QOS_POLICY_NAME
 - TYPESUPPORT, 1343
- DDS_TypeSupportQosPolicy, 2081
 - cdr_padding_kind, 2082
 - plugin_data, 2082
- DDS_UInt8
 - DDS-Specific Primitive Types, 1202
- DDS_UInt8Seq, 2083
- DDS_UNICAST_ONLY_TRANSPORT_MULTICAST_QOS
 - TRANSPORT_MULTICAST, 1336
- DDS_UnionMember, 2083
 - is_pointer, 2084
 - labels, 2084
 - name, 2084
 - type, 2085
- DDS_UnionMemberSeq, 2085
- DDS_UNKNOWN_SAMPLE_IDENTITY
 - WriteParams, 1392
- DDS_UNKNOWN_SERVICE_REQUEST_ID
 - ServiceRequest Built-in Topic, 1100
- DDS_UNREGISTERED_INSTANCE_REPLACEMENT
 - DATA_WRITER_RESOURCE_LIMITS, 1270
- DDS_UnsignedLong
 - DDS-Specific Primitive Types, 1203
- DDS_UnsignedLongLong

- DDS-Specific Primitive Types, 1203
- DDS_UnsignedLongLongSeq, 2085
- DDS_UnsignedLongSeq, 2086
- DDS_UnsignedShort
 - DDS-Specific Primitive Types, 1202
- DDS_UnsignedShortSeq, 2086
- DDS_USER_EXCEPTION_CODE
 - Exception Codes, 1219
- DDS_USERDATA_QOS_POLICY_ID
 - All QoS Policies, 1246
- DDS_USERDATA_QOS_POLICY_NAME
 - USER_DATA, 1343
- DDS_UserDataQosPolicy, 2086
 - value, 2087
- DDS_ValueMember, 2088
 - access, 2089
 - bits, 2089
 - id, 2090
 - is_key, 2089
 - is_optional, 2090
 - is_pointer, 2089
 - name, 2089
 - type, 2089
- DDS_ValueMemberSeq, 2090
- DDS_ValueModifier
 - Type Code Support, 242
- DDS_VENDOR_ID_LENGTH_MAX
 - Common types and functions, 1105
- DDS_VendorEndpointTrustAttributesMask
 - Built-in Topic's Trust Types, 311
- DDS_VendorId_t, 2090
 - vendorId, 2091
- DDS_ViewStateKind
 - View States, 712
- DDS_ViewStateMask
 - View States, 712
- DDS_VirtualWriterInfo, 2091
- DDS_Visibility
 - Type Code Support, 242
- DDS_VM_ABSTRACT
 - Type Code Support, 239
- DDS_VM_CUSTOM
 - Type Code Support, 238
- DDS_VM_NONE
 - Type Code Support, 238
- DDS_VM_TRUNCATABLE
 - Type Code Support, 239
- DDS_VOLATILE_DURABILITY_QOS
 - DURABILITY, 1287
- DDS_WaitSet
 - Conditions and WaitSets, 1371
- DDS_WaitSet_attach_condition
 - Conditions and WaitSets, 1383
- DDS_WaitSet_delete
 - Conditions and WaitSets, 1381
- DDS_WaitSet_detach_condition
 - Conditions and WaitSets, 1384
- DDS_WaitSet_get_conditions
 - Conditions and WaitSets, 1384
- DDS_WaitSet_get_property
 - Conditions and WaitSets, 1382
- DDS_WaitSet_new
 - Conditions and WaitSets, 1380
- DDS_WaitSet_new_ex
 - Conditions and WaitSets, 1380
- DDS_WaitSet_set_property
 - Conditions and WaitSets, 1382
- DDS_WaitSet_wait
 - Conditions and WaitSets, 1382
- DDS_WaitSetProperty_t, 2091
 - max_event_count, 2092
 - max_event_delay, 2092
- DDS_WaitSetProperty_t_INITIALIZER
 - Conditions and WaitSets, 1369
- DDS_WAL_PERSISTENT_JOURNAL
 - DURABILITY, 1286
- DDS_Wchar
 - DDS-Specific Primitive Types, 1202
- DDS_WcharSeq, 2093
- DDS_WEB_INTEGRATION_SERVICE_QOS
 - SERVICE, 1327
- DDS_WIREPROTOCOL_QOS_POLICY_ID
 - All QoS Policies, 1247
- DDS_WIREPROTOCOL_QOS_POLICY_NAME
 - WIRE_PROTOCOL, 1349
- DDS_WireProtocolQosPolicy, 2093
 - check_crc, 2100
 - compute_crc, 2100
 - participant_id, 2097
 - rtps_app_id, 2098
 - rtps_auto_id_kind, 2100
 - rtps_host_id, 2097
 - rtps_instance_id, 2098
 - rtps_reserved_port_mask, 2099
 - rtps_well_known_ports, 2099
- DDS_WireProtocolQosPolicyAutoKind
 - WIRE_PROTOCOL, 1348
- DDS_WRITEPARAMS_DEFAULT
 - WriteParams, 1392
- DDS_WriteParams_reset
 - WriteParams, 1391
- DDS_WriteParams_t, 2101
 - cookie, 2103
 - flag, 2104
 - handle, 2103
 - identity, 2102
 - priority, 2103
 - related_reader_guid, 2105

- related_sample_identity, 2102
- related_source_guid, 2105
- replace_auto, 2102
- source_guid, 2104
- source_timestamp, 2103
- DDS_WRITER_REMOVED_BATCH_SAMPLE
 - Sample Flags, 1388
- DDS_WRITERDATA_LIFECYCLE_QOS_POLICY_ID
 - All QoS Policies, 1247
- DDS_WRITERDATA_LIFECYCLE_QOS_POLICY_NAME
 - WRITER_DATA_LIFECYCLE, 1344
- DDS_WriterDataLifecycleQosPolicy, 2106
 - autodispose_unregistered_instances, 2107
 - autopurge_disposed_instances_delay, 2108
 - autopurge_unregistered_instances_delay, 2107
- DDS_Wstring_alloc
 - String Support, 1530
- DDS_Wstring_copy
 - String Support, 1531
- DDS_Wstring_copy_and_widen
 - String Support, 1531
- DDS_Wstring_dup
 - String Support, 1532
- DDS_Wstring_dup_and_widen
 - String Support, 1532
- DDS_Wstring_free
 - String Support, 1532
- DDS_Wstring_length
 - String Support, 1530
- DDS_WstringSeq, 2109
- DDS_XCDR2_DATA_REPRESENTATION
 - DATA_REPRESENTATION, 1258
- DDS_XCDR_DATA_REPRESENTATION
 - DATA_REPRESENTATION, 1257
- DDS_XML_DATA_REPRESENTATION
 - DATA_REPRESENTATION, 1257
- DDS_XML_PRINT_FORMAT
 - Topics, 186
- DDS_ZERO_CDR_PADDING
 - TYPESUPPORT, 1342
- DEADLINE, 1272
 - DDS_DEADLINE_QOS_POLICY_NAME, 1273
- deadline
 - DDS_DataReaderQos, 1613
 - DDS_DataWriterQos, 1668
 - DDS_PublicationBuiltinTopicData, 1908
 - DDS_SubscriptionBuiltinTopicData, 2015
 - DDS_TopicBuiltinTopicData, 2037
 - DDS_TopicQos, 2044
- Debuggability, 847
- dedicated_participant
 - DDS_MonitoringDistributionSettings, 1839
- default_domain_announcement_period
 - DDS_DiscoveryConfigQosPolicy, 1704
- default_unicast
 - DDS_DomainParticipantQos, 1729
- default_unicast_locators
 - DDS_ParticipantBuiltinTopicData, 1873
- delete_optional_members
 - DDS_TypeDeallocationParams_t, 2080
- delete_pointers
 - DDS_TypeDeallocationParams_t, 2080
- delete_thread
 - DDS_ThreadFactory, 2028
- deny_interfaces_list
 - NDDS_Transport_Property_t, 2126
- deny_interfaces_list_length
 - NDDS_Transport_Property_t, 2126
- deny_multicast_interfaces_list
 - NDDS_Transport_Property_t, 2127
- deny_multicast_interfaces_list_length
 - NDDS_Transport_Property_t, 2127
- Deprecated QoS, 965
- depth
 - DDS_HistoryQosPolicy, 1807
- deserialized_type_object_dynamic_allocation_threshold
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1751
- DESTINATION_ORDER, 1273
 - DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS, 1274
 - DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS, 1274
 - DDS_DESTINATIONORDER_QOS_POLICY_NAME, 1275
 - DDS_DestinationOrderQosPolicyKind, 1274
 - DDS_DestinationOrderQosPolicyScopeKind, 1275
 - DDS_INSTANCE_SCOPE_DESTINATIONORDER_QOS, 1275
 - DDS_TOPIC_SCOPE_DESTINATIONORDER_QOS, 1275
- destination_order
 - DDS_DataReaderQos, 1614
 - DDS_DataWriterQos, 1669
 - DDS_PublicationBuiltinTopicData, 1909
 - DDS_SubscriptionBuiltinTopicData, 2016
 - DDS_TopicBuiltinTopicData, 2038
 - DDS_TopicQos, 2044
- detached_instance_count
 - DDS_DataReaderCacheStatus, 1592
- detached_instance_count_peak
 - DDS_DataReaderCacheStatus, 1592
- direct_communication
 - DDS_DurabilityQosPolicy, 1762
- disable_asynchronous_batch
 - DDS_AsynchronousPublisherQosPolicy, 1541
- disable_asynchronous_write
 - DDS_AsynchronousPublisherQosPolicy, 1540

- disable_fragmentation_support
 - DDS_BuiltinTopicReaderResourceLimits_t, 1560
 - DDS_DataReaderResourceLimitsQosPolicy, 1626
- disable_inline_keyhash
 - DDS_DataWriterProtocolQosPolicy, 1651
- disable_interface_tracking
 - NDDS_Transport_UDPv4_Property_t, 2142
 - NDDS_Transport_UDPv4_WAN_Property_t, 2150
 - NDDS_Transport_UDPv6_Property_t, 2162
- disable_positive_acks
 - DDS_DataReaderProtocolQosPolicy, 1598
 - DDS_DataWriterProtocolQosPolicy, 1651
 - DDS_PublicationBuiltinTopicData, 1912
 - DDS_SubscriptionBuiltinTopicData, 2020
- disable_positive_acks_decrease_sample_keep_duration_factor
 - DDS_RtpsReliableWriterProtocol_t, 1972
- disable_positive_acks_enable_adaptive_sample_keep_duration
 - DDS_RtpsReliableWriterProtocol_t, 1971
- disable_positive_acks_increase_sample_keep_duration_factor
 - DDS_RtpsReliableWriterProtocol_t, 1972
- disable_positive_acks_max_sample_keep_duration
 - DDS_RtpsReliableWriterProtocol_t, 1971
- disable_positive_acks_min_sample_keep_duration
 - DDS_RtpsReliableWriterProtocol_t, 1970
- disable_repair_piggyback_heartbeat
 - DDS_RtpsReliableWriterProtocol_t, 1978
- disable_topic_query_publication
 - DDS_AsynchronousPublisherQosPolicy, 1542
- disabled_metrics_selection
 - DDS_MonitoringMetricSelection, 1850
- DISCOVERY, 1275
 - DDS_DISCOVERY_QOS_POLICY_NAME, 1276
- Discovery, 853
- discovery
 - DDS_DomainParticipantQos, 1729
- Discovery Isolation, 858
- Discovery Metadata, 958
- DISCOVERY_CONFIG, 1276
 - DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK, 1279
 - DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_DEFAULT, 1278
 - DDS_DISCOVERYCONFIG_BUILTIN_CHANNEL_MASK_NONE, 1278
 - DDS_DISCOVERYCONFIG_BUILTIN_DPSE, 1281
 - DDS_DISCOVERYCONFIG_BUILTIN_PLUGIN_MASK_DEFAULT, 1278
 - DDS_DISCOVERYCONFIG_BUILTIN_PLUGIN_MASK_NONE, 1278
 - DDS_DISCOVERYCONFIG_BUILTIN_SDP, 1280
 - DDS_DISCOVERYCONFIG_BUILTIN_SDP2, 1281
 - DDS_DISCOVERYCONFIG_BUILTIN_SEDP, 1280
 - DDS_DISCOVERYCONFIG_BUILTIN_SPDP, 1280
 - DDS_DISCOVERYCONFIG_BUILTIN_SPDP2, 1281
- DDS_DISCOVERYCONFIG_QOS_POLICY_NAME, 1283
- DDS_DISCOVERYCONFIG_SERVICE_REQUEST_CHANNEL, 1281
- DDS_DISCOVERYCONFIG_TYPE_LOOKUP_SERVICE_CHANNEL, 1281
- DDS_DiscoveryConfigBuiltinChannelKind, 1281
- DDS_DiscoveryConfigBuiltinChannelKindMask, 1279
- DDS_DiscoveryConfigBuiltinPluginKind, 1280
- DDS_DiscoveryConfigBuiltinPluginKindMask, 1279
- DDS_LIVELINESS_BASED_REMOTE_PARTICIPANT_PURGE, 1282
- DDS_NO_REMOTE_PARTICIPANT_PURGE, 1283
- DDS_RemoteParticipantPurgeKind, 1282
- discovery_config
 - DDS_DomainParticipantQos, 1730
- disposed_generation_count
 - DDS_SampleInfo, 1990
- disposed_instance_count
 - DDS_DataReaderCacheStatus, 1591
 - DDS_DataWriterCacheStatus, 1643
- disposed_instance_count_peak
 - DDS_DataReaderCacheStatus, 1591
 - DDS_DataWriterCacheStatus, 1643
- disposed_instance_removal
 - DDS_DataReaderResourceLimitsInstanceReplacementSettings, 1618
- distribution_settings
 - DDS_MonitoringQosPolicy, 1854
- dll
 - DDS_TransportMulticastMappingFunction_t, 2055
- dns_tracker_polling_period
 - DDS_DiscoveryConfigQosPolicy, 1711
- Documentation Roadmap, 1010
- Domain Module, 26
- domain_entity_qos_library_name
 - DDS_DomainParticipantConfigParams_t, 1720
- domain_entity_qos_profile_name
 - DDS_DomainParticipantConfigParams_t, 1721
- DOMAIN_ID_GAIN, 1284
 - DDS_DomainParticipantConfigParams_t, 1720
- DDS_MonitoringDedicatedParticipantSettings, 1837
- DDS_ParticipantBuiltinTopicData, 1873
- domain_id_gain
 - DDS_RtpsWellKnownPorts_t, 1981
- DOMAIN_PARTICIPANT_RESOURCE_LIMITS, 1283
- DDS_AUTO_COUNT, 1284
 - DDS_DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_NAME, 1285
- DDS_DomainParticipantResourceLimitsIgnoredEntityReplacementKind, 1284
- DDS_NO_REPLACEMENT_IGNORED_ENTITY_REPLACEMENT, 1284

- DDS_NOT_ALIVE_FIRST_IGNORED_ENTITY_REPLACEMENT, 1284
- DomainParticipantConfigParams, 1430
 - DDS_DOMAIN_ID_USE_XML_CONFIG, 1431
 - DDS_DomainParticipantConfigParams_t_INITIALIZER, 1431
 - DDS_ENTITY_NAME_USE_XML_CONFIG, 1431
 - DDS_QOS_ELEMENT_NAME_USE_XML_CONFIG, 1431
- DomainParticipantFactory, 27
 - DDS_DomainParticipantFactory, 32
 - DDS_DomainParticipantFactory_create_participant, 41
 - DDS_DomainParticipantFactory_create_participant_from_config, 59
 - DDS_DomainParticipantFactory_create_participant_from_config_w_params, 60
 - DDS_DomainParticipantFactory_create_participant_with_profile, 43
 - DDS_DomainParticipantFactory_delete_participant, 44
 - DDS_DomainParticipantFactory_finalize_instance, 38
 - DDS_DomainParticipantFactory_get_datareader_qos_from_profile, 54
 - DDS_DomainParticipantFactory_get_datareader_qos_from_profile_w_topic_name, 54
 - DDS_DomainParticipantFactory_get_datawriter_qos_from_profile, 55
 - DDS_DomainParticipantFactory_get_datawriter_qos_from_profile_w_topic_name, 55
 - DDS_DomainParticipantFactory_get_default_library, 49
 - DDS_DomainParticipantFactory_get_default_participant_qos, 41
 - DDS_DomainParticipantFactory_get_default_profile, 50
 - DDS_DomainParticipantFactory_get_default_profile_library, 51
 - DDS_DomainParticipantFactory_get_instance, 38
 - DDS_DomainParticipantFactory_get_participant_factory_qos, 51
 - DDS_DomainParticipantFactory_get_participant_qos_from_profile, 52
 - DDS_DomainParticipantFactory_get_participants, 63
 - DDS_DomainParticipantFactory_get_publisher_qos_from_profile, 52
 - DDS_DomainParticipantFactory_get_qos, 45
 - DDS_DomainParticipantFactory_get_qos_profile_libraries, 57
 - DDS_DomainParticipantFactory_get_qos_profiles, 58
 - DDS_DomainParticipantFactory_get_subscriber_qos_from_profile, 53
- DDS_DomainParticipantFactory_get_topic_qos_from_profile, 56
- DDS_DomainParticipantFactory_get_topic_qos_from_profile_w_topic_name, 57
- DDS_DomainParticipantFactory_get_typecode_from_config, 59
- DDS_DomainParticipantFactory_load_profiles, 46
- DDS_DomainParticipantFactory_lookup_participant, 45
- DDS_DomainParticipantFactory_lookup_participant_by_name, 61
- DDS_DomainParticipantFactory_register_type_support, 62
- DDS_DomainParticipantFactory_RegisterTypeFunction, 33
- DDS_DomainParticipantFactory_reload_profiles, 47
- DDS_DomainParticipantFactory_set_default_library, 49
- DDS_DomainParticipantFactory_set_default_participant_qos, 39
- DDS_DomainParticipantFactory_set_default_participant_qos_with_profile, 40
- DDS_DomainParticipantFactory_set_default_profile, 49
- DDS_DomainParticipantFactory_set_qos, 46
- DDS_DomainParticipantFactory_set_thread_factory, 63
- DDS_DomainParticipantFactory_unload_profiles, 47
- DDS_DomainParticipantFactory_unregister_thread, 58
- DDS_DomainParticipantFactoryQos_copy, 37
- DDS_DomainParticipantFactoryQos_equals, 33
- DDS_DomainParticipantFactoryQos_finalize, 36
- DDS_DomainParticipantFactoryQos_initialize, 36
- DDS_DomainParticipantFactoryQos_INITIALIZER, 31
- DDS_DomainParticipantFactoryQos_print, 34
- DDS_DomainParticipantFactoryQos_to_string, 34
- DDS_DomainParticipantFactoryQos_to_string_w_params, 35
- DDS_DOMAIN_PARTICIPANT_CONFIG_PARAMS_DEFAULT, 65
- DDS_DOMAIN_PARTICIPANT_QOS_DEFAULT, 65
- DDS_TheParticipantFactory, 32
- DomainParticipants, 66
- DDS_DomainId_t, 77
- DDS_DomainParticipant, 77
- DDS_DomainParticipant_add_peer, 149
- DDS_DomainParticipant_as_entity, 83
- DDS_DomainParticipant_assert_liveliness, 136
- DDS_DomainParticipant_banish_ignored_participants, 134
- DDS_DomainParticipant_contains_entity, 139
- DDS_DomainParticipant_create_contentfilteredtopic, 139

- 118
- DDS_DomainParticipant_create_contentfilteredtopic_with_filters, 119
- DDS_DomainParticipant_create_datareader, 112
- DDS_DomainParticipant_create_datareader_with_profile, 113
- DDS_DomainParticipant_create_datawriter, 108
- DDS_DomainParticipant_create_datawriter_with_profile, 110
- DDS_DomainParticipant_create_flowcontroller, 124
- DDS_DomainParticipant_create_multitopic, 122
- DDS_DomainParticipant_create_publisher, 103
- DDS_DomainParticipant_create_publisher_with_profile, 104
- DDS_DomainParticipant_create_subscriber, 106
- DDS_DomainParticipant_create_subscriber_with_profile, 107
- DDS_DomainParticipant_create_topic, 115
- DDS_DomainParticipant_create_topic_with_profile, 116
- DDS_DomainParticipant_delete_contained_entities, 135
- DDS_DomainParticipant_delete_contentfilteredtopic, 119
- DDS_DomainParticipant_delete_datareader, 114
- DDS_DomainParticipant_delete_datawriter, 111
- DDS_DomainParticipant_delete_durable_subscription, 141
- DDS_DomainParticipant_delete_flowcontroller, 124
- DDS_DomainParticipant_delete_multitopic, 123
- DDS_DomainParticipant_delete_publisher, 105
- DDS_DomainParticipant_delete_subscriber, 108
- DDS_DomainParticipant_delete_topic, 117
- DDS_DomainParticipant_find_topic, 126
- DDS_DomainParticipant_get_builtin_subscriber, 129
- DDS_DomainParticipant_get_current_time, 138
- DDS_DomainParticipant_get_default_datareader_qos, 96
- DDS_DomainParticipant_get_default_datawriter_qos, 91
- DDS_DomainParticipant_get_default_flowcontroller_property, 98
- DDS_DomainParticipant_get_default_library, 100
- DDS_DomainParticipant_get_default_profile, 100
- DDS_DomainParticipant_get_default_profile_library, 101
- DDS_DomainParticipant_get_default_publisher_qos, 89
- DDS_DomainParticipant_get_default_subscriber_qos, 94
- DDS_DomainParticipant_get_default_topic_qos, 86
- DDS_DomainParticipant_get_discovered_participant_data, 145
- DDS_DomainParticipant_get_discovered_participant_subject_name, 146
- DDS_DomainParticipant_get_discovered_participants, 144
- DDS_DomainParticipant_get_discovered_participants_from_subject_name, 144
- DDS_DomainParticipant_get_discovered_topic_data, 147
- DDS_DomainParticipant_get_discovered_topics, 147
- DDS_DomainParticipant_get_dns_tracker_polling_period, 143
- DDS_DomainParticipant_get_domain_id, 135
- DDS_DomainParticipant_get_implicit_publisher, 129
- DDS_DomainParticipant_get_implicit_subscriber, 130
- DDS_DomainParticipant_get_listener, 154
- DDS_DomainParticipant_get_listenerX, 156
- DDS_DomainParticipant_get_participant_protocol_status, 128
- DDS_DomainParticipant_get_publishers, 137
- DDS_DomainParticipant_get_qos, 152
- DDS_DomainParticipant_get_subscribers, 138
- DDS_DomainParticipant_get_typecode, 125
- DDS_DomainParticipant_ignore_participant, 131
- DDS_DomainParticipant_ignore_publication, 132
- DDS_DomainParticipant_ignore_subscription, 133
- DDS_DomainParticipant_ignore_topic, 132
- DDS_DomainParticipant_lookup_contentfilter, 121
- DDS_DomainParticipant_lookup_datareader_by_name, 159
- DDS_DomainParticipant_lookup_datawriter_by_name, 158
- DDS_DomainParticipant_lookup_flowcontroller, 128
- DDS_DomainParticipant_lookup_publisher_by_name, 156
- DDS_DomainParticipant_lookup_subscriber_by_name, 157
- DDS_DomainParticipant_lookup_topicdescription, 127
- DDS_DOMAINPARTICIPANT_QOS_PRINT_ALL, 160
- DDS_DomainParticipant_register_contentfilter, 120
- DDS_DomainParticipant_register_durable_subscription, 139
- DDS_DomainParticipant_remove_peer, 150
- DDS_DomainParticipant_resume_endpoint_discovery, 141
- DDS_DomainParticipant_set_default_datareader_qos, 97
- DDS_DomainParticipant_set_default_datareader_qos_with_profile, 97
- DDS_DomainParticipant_set_default_datawriter_qos, 92
- DDS_DomainParticipant_set_default_datawriter_qos_with_profile, 92

- 93
- DDS_DomainParticipant_set_default_flowcontroller_property, 99
- DDS_DomainParticipant_set_default_library, 101
- DDS_DomainParticipant_set_default_profile, 102
- DDS_DomainParticipant_set_default_publisher_qos, 89
- DDS_DomainParticipant_set_default_publisher_qos_with_profile, 90
- DDS_DomainParticipant_set_default_subscriber_qos, 94
- DDS_DomainParticipant_set_default_subscriber_qos_with_profile, 95
- DDS_DomainParticipant_set_default_topic_qos, 87
- DDS_DomainParticipant_set_default_topic_qos_with_profile, 88
- DDS_DomainParticipant_set_dns_tracker_polling_period, 142
- DDS_DomainParticipant_set_listener, 154
- DDS_DomainParticipant_set_property, 153
- DDS_DomainParticipant_set_qos, 151
- DDS_DomainParticipant_set_qos_with_profile, 152
- DDS_DomainParticipant_take_discovery_snapshot, 148
- DDS_DomainParticipant_unregister_contentfilter, 121
- DDS_DOMAINPARTICIPANTFACTORY_QOS_PRINT_ALL, 160
- DDS_DomainParticipantListener_INITIALIZER, 75
- DDS_DomainParticipantListener_InvalidLocalAdvanceNoticeStatus_equals, 77
- DDS_DomainParticipantProtocolStatus_copy, 84
- DDS_DomainParticipantProtocolStatus_equals, 86
- DDS_DomainParticipantProtocolStatus_finalize, 85
- DDS_DomainParticipantProtocolStatus_initialize, 84
- DDS_DomainParticipantProtocolStatus_INITIALIZER, 76
- DDS_DomainParticipantQos_copy, 83
- DDS_DomainParticipantQos_equals, 79
- DDS_DomainParticipantQos_finalize, 82
- DDS_DomainParticipantQos_initialize, 81
- DDS_DomainParticipantQos_INITIALIZER, 75
- DDS_DomainParticipantQos_print, 79
- DDS_DomainParticipantQos_to_string, 80
- DDS_DomainParticipantQos_to_string_w_params, 81
- DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT, 162
- DDS_INVALID_LOCAL_ADVANCE_NOTICE_STATUS_EXPIRATION_INTERVAL, 75
- DDS_InvalidLocalAdvanceNoticeStatus_equals, 79
- DDS_PUBLISHER_QOS_DEFAULT, 161
- DDS_PUBLISHER_QOS_PRINT_ALL, 163
- DDS_SQLFILTER_NAME, 163
- DDS_STRINGMATCHFILTER_NAME, 164
- DDS_SUBSCRIBER_QOS_DEFAULT, 162
- DDS_SUBSCRIBER_QOS_PRINT_ALL, 164
- DDS_TOPIC_QOS_DEFAULT, 160
- DDS_TOPIC_QOS_PRINT_ALL, 164
- drop_incomplete_coherent_set
- DDS_PresentationQosPolicy, 1894
- drop_profile_content
- NDDS_Utility_NetworkCaptureParams_t, 2166
- dropped_fragment_count
- DDS_DataReaderProtocolStatus, 1610
- duplicate_sample_bytes
- DDS_DataReaderProtocolStatus, 1604
- duplicate_sample_bytes_change
- DDS_DataReaderProtocolStatus, 1604
- duplicate_sample_count
- DDS_DataReaderProtocolStatus, 1604
- duplicate_sample_count_change
- DDS_DataReaderProtocolStatus, 1604
- DURABILITY, 1285
- DDS_AUTO_WRITER_DEPTH, 1288
- DDS_DELETE_PERSISTENT_JOURNAL, 1286
- DDS_DURABILITY_QOS_POLICY_NAME, 1288
- DDS_DurabilityQosPolicyKind, 1287
- DDS_FULL_PERSISTENT_SYNCHRONIZATION, 1287
- DDS_MEMORY_PERSISTENT_JOURNAL, 1286
- DDS_NORMAL_PERSISTENT_SYNCHRONIZATION, 1287
- DDS_OFF_PERSISTENT_JOURNAL, 1286
- DDS_OFF_PERSISTENT_SYNCHRONIZATION, 1287
- DDS_PERSIST_PERSISTENT_JOURNAL, 1286
- DDS_PERSISTENT_DURABILITY_QOS, 1288
- DDS_PersistentJournalKind, 1286
- DDS_PersistentSynchronizationKind, 1286
- DDS_TRANSIENT_DURABILITY_QOS, 1288
- DDS_TRANSIENT_LOCAL_DURABILITY_QOS, 1288
- DDS_TRUNCATE_PERSISTENT_JOURNAL, 1286
- DDS_VOLATILE_DURABILITY_QOS, 1287
- DDS_WAL_PERSISTENT_JOURNAL, 1286
- Durability, 860
- durability
- DDS_DataReaderQos, 1613
- DDS_DataWriterQos, 1668
- DDS_PublicationBuiltinTopicData, 1907
- DDS_SubscriptionBuiltinTopicData, 2015
- DDS_OpinionBuiltinTopicData, 2036
- DDS_TopicQos, 2043
- Durability and Persistence, 777
- DURABILITY_SERVICE, 1289
- DDS_DURABILITYSERVICE_QOS_POLICY_NAME, 1289

- durability_service
 - DDS_DataWriterQos, 1668
 - DDS_PublicationBuiltinTopicData, 1908
 - DDS_TopicBuiltinTopicData, 2037
 - DDS_TopicQos, 2044
- duration
 - DDS_LatencyBudgetQosPolicy, 1815
 - DDS_LifespanQosPolicy, 1816
- Dynamic Data, 313
 - DDS_DYNAMIC_DATA_JSON_PARSER_PROPERTIES_DEFAULT, 438
 - DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED, 324
 - DDS_DYNAMIC_DATA_PROPERTY_DEFAULT, 437
 - DDS_DYNAMIC_DATA_TYPE_PROPERTY_DEFAULT, 437
 - DDS_DynamicData_bind_complex_member, 342
 - DDS_DynamicData_bind_type, 340
 - DDS_DynamicData_clear_all_members, 330
 - DDS_DynamicData_clear_member, 331
 - DDS_DynamicData_clear_optional_member, 330
 - DDS_DynamicData_copy, 329
 - DDS_DynamicData_delete, 329
 - DDS_DynamicData_equal, 329
 - DDS_DynamicData_finalize, 328
 - DDS_DynamicData_from_cdr_buffer, 337
 - DDS_DynamicData_from_string, 339
 - DDS_DynamicData_get_boolean, 356
 - DDS_DynamicData_get_boolean_array, 372
 - DDS_DynamicData_get_boolean_seq, 386
 - DDS_DynamicData_get_cdr_buffer, 332
 - DDS_DynamicData_get_char, 356
 - DDS_DynamicData_get_char_array, 373
 - DDS_DynamicData_get_char_seq, 387
 - DDS_DynamicData_get_complex_member, 365
 - DDS_DynamicData_get_discriminator, 335
 - DDS_DynamicData_get_double, 355
 - DDS_DynamicData_get_double_array, 371
 - DDS_DynamicData_get_double_seq, 385
 - DDS_DynamicData_get_float, 354
 - DDS_DynamicData_get_float_array, 370
 - DDS_DynamicData_get_float_seq, 384
 - DDS_DynamicData_get_info, 340
 - DDS_DynamicData_get_int8, 361
 - DDS_DynamicData_get_int8_array, 379
 - DDS_DynamicData_get_int8_seq, 392
 - DDS_DynamicData_get_long, 351
 - DDS_DynamicData_get_long_array, 366
 - DDS_DynamicData_get_long_seq, 381
 - DDS_DynamicData_get_longdouble, 360
 - DDS_DynamicData_get_longdouble_array, 377
 - DDS_DynamicData_get_longdouble_seq, 390
 - DDS_DynamicData_get_longlong, 358
 - DDS_DynamicData_get_longlong_array, 375
 - DDS_DynamicData_get_longlong_seq, 389
 - DDS_DynamicData_get_member_count, 345
 - DDS_DynamicData_get_member_info, 347
 - DDS_DynamicData_get_member_info_by_index, 348
 - DDS_DynamicData_get_member_type, 349
 - DDS_DynamicData_get_octet, 357
 - DDS_DynamicData_get_octet_array, 374
 - DDS_DynamicData_get_octet_seq, 388
 - DDS_DynamicData_get_short, 351
 - DDS_DynamicData_get_short_array, 367
 - DDS_DynamicData_get_short_seq, 382
 - DDS_DynamicData_get_string, 363
 - DDS_DynamicData_get_type, 344
 - DDS_DynamicData_get_type_kind, 345
 - DDS_DynamicData_get_uint8, 362
 - DDS_DynamicData_get_uint8_array, 380
 - DDS_DynamicData_get_uint8_seq, 393
 - DDS_DynamicData_get_ulong, 352
 - DDS_DynamicData_get_ulong_array, 368
 - DDS_DynamicData_get_ulong_seq, 383
 - DDS_DynamicData_get_ulonglong, 359
 - DDS_DynamicData_get_ulonglong_array, 376
 - DDS_DynamicData_get_ulonglong_seq, 389
 - DDS_DynamicData_get_ushort, 353
 - DDS_DynamicData_get_ushort_array, 369
 - DDS_DynamicData_get_ushort_seq, 383
 - DDS_DynamicData_get_wchar, 361
 - DDS_DynamicData_get_wchar_array, 378
 - DDS_DynamicData_get_wchar_seq, 391
 - DDS_DynamicData_get_wstring, 364
 - DDS_DynamicData_initialize, 327
 - DDS_DynamicData_is_cdr, 333
 - DDS_DynamicData_is_member_key, 350
 - DDS_DynamicData_member_exists, 345
 - DDS_DynamicData_member_exists_in_type, 346
 - DDS_DynamicData_new, 327
 - DDS_DynamicData_print, 336
 - DDS_DynamicData_set_boolean, 398
 - DDS_DynamicData_set_boolean_array, 413
 - DDS_DynamicData_set_boolean_seq, 425
 - DDS_DynamicData_set_cdr_buffer, 332
 - DDS_DynamicData_set_char, 399
 - DDS_DynamicData_set_char_array, 414
 - DDS_DynamicData_set_char_seq, 426
 - DDS_DynamicData_set_complex_member, 406
 - DDS_DynamicData_set_discriminator, 334
 - DDS_DynamicData_set_double, 397
 - DDS_DynamicData_set_double_array, 412
 - DDS_DynamicData_set_double_seq, 424
 - DDS_DynamicData_set_float, 396
 - DDS_DynamicData_set_float_array, 411
 - DDS_DynamicData_set_float_seq, 424
 - DDS_DynamicData_set_int8, 403

- DDS_DynamicData_set_int8_array, 419
- DDS_DynamicData_set_int8_seq, 430
- DDS_DynamicData_set_long, 394
- DDS_DynamicData_set_long_array, 408
- DDS_DynamicData_set_long_seq, 421
- DDS_DynamicData_set_longdouble, 402
- DDS_DynamicData_set_longdouble_array, 417
- DDS_DynamicData_set_longdouble_seq, 429
- DDS_DynamicData_set_longlong, 400
- DDS_DynamicData_set_longlong_array, 415
- DDS_DynamicData_set_longlong_seq, 427
- DDS_DynamicData_set_octet, 399
- DDS_DynamicData_set_octet_array, 415
- DDS_DynamicData_set_octet_seq, 427
- DDS_DynamicData_set_short, 394
- DDS_DynamicData_set_short_array, 408
- DDS_DynamicData_set_short_seq, 421
- DDS_DynamicData_set_string, 405
- DDS_DynamicData_set_uint8, 404
- DDS_DynamicData_set_uint8_array, 420
- DDS_DynamicData_set_uint8_seq, 431
- DDS_DynamicData_set_ulong, 395
- DDS_DynamicData_set_ulong_array, 409
- DDS_DynamicData_set_ulong_seq, 422
- DDS_DynamicData_set_ulonglong, 401
- DDS_DynamicData_set_ulonglong_array, 416
- DDS_DynamicData_set_ulonglong_seq, 428
- DDS_DynamicData_set_ushort, 396
- DDS_DynamicData_set_ushort_array, 410
- DDS_DynamicData_set_ushort_seq, 423
- DDS_DynamicData_set_wchar, 402
- DDS_DynamicData_set_wchar_array, 418
- DDS_DynamicData_set_wchar_seq, 430
- DDS_DynamicData_set_wstring, 405
- DDS_DynamicData_to_cdr_buffer, 338
- DDS_DynamicData_to_cdr_buffer_ex, 338
- DDS_DynamicData_to_string, 339
- DDS_DynamicData_unbind_complex_member, 344
- DDS_DynamicData_unbind_type, 341
- DDS_DynamicDataMemberId, 325
- DDS_DynamicDataProperty_t_INITIALIZER, 324
- DDS_DynamicDataReader, 326
- DDS_DynamicDataTypeProperty_t_INITIALIZER, 325
- DDS_DynamicDataTypeSupport, 326
- DDS_DynamicDataTypeSupport_copy_data, 436
- DDS_DynamicDataTypeSupport_create_data, 435
- DDS_DynamicDataTypeSupport_delete, 433
- DDS_DynamicDataTypeSupport_delete_data, 435
- DDS_DynamicDataTypeSupport_finalize_data, 436
- DDS_DynamicDataTypeSupport_get_data_type, 434
- DDS_DynamicDataTypeSupport_get_type_name, 434
- DDS_DynamicDataTypeSupport_initialize_data, 436
- DDS_DynamicDataTypeSupport_new, 432
- DDS_DynamicDataTypeSupport_print_data, 435
- DDS_DynamicDataTypeSupport_register_type, 433
- DDS_DynamicDataTypeSupport_unregister_type, 433
- DDS_DynamicDataWriter, 325
- dynamically_allocate_fragmented_samples
 - DDS_BuiltinTopicReaderResourceLimits_t, 1563
 - DDS_DataReaderResourceLimitsQosPolicy, 1628
- element_count
 - DDS_DynamicDataMemberInfo, 1779
- element_kind
 - DDS_DynamicDataMemberInfo, 1779
- empty_reliable_writer_cache
 - DDS_ReliableWriterCacheChangedStatus, 1943
- enable
 - DDS_BatchQosPolicy, 1552
 - DDS_MonitoringDedicatedParticipantSettings, 1836
 - DDS_MonitoringQosPolicy, 1854
 - DDS_PersistentStorageSettings, 1885
 - DDS_TopicQueryDispatchQosPolicy, 2048
- enable_data_consistency_check
 - DDS_DataWriterShmemRefTransferModeSettings, 1682
- enable_endpoint_discovery
 - DDS_DiscoveryQosPolicy, 1718
- enable_multicast_periodic_heartbeat
 - DDS_RtpsReliableWriterProtocol_t, 1977
- enable_required_subscriptions
 - DDS_AvailabilityQosPolicy, 1550
- enable_udp_debugging
 - NDDS_Transport_Shmem_Property_t, 2131
- enable_v4mapped
 - NDDS_Transport_UDPv6_Property_t, 2159
- enabled_builtin_channels
 - DDS_DiscoveryConfigQosPolicy, 1701
- enabled_metrics_selection
 - DDS_MonitoringMetricSelection, 1850
- enabled_transports
 - DDS_DiscoveryQosPolicy, 1716
 - DDS_TransportSelectionQosPolicy, 2065
- endpoint_type_object_lb_serialization_threshold
 - DDS_DiscoveryConfigQosPolicy, 1711
- Entity Support, 1359
 - DDS_DomainEntity, 1363
 - DDS_Entity, 1360
 - DDS_Entity_enable, 1363
 - DDS_Entity_get_entity_kind, 1366
 - DDS_Entity_get_instance_handle, 1366
 - DDS_Entity_get_status_changes, 1365
 - DDS_Entity_get_statuscondition, 1365
 - DDS_Listener_INITIALIZER, 1360
- Entity Use Cases, 988

- ENTITY_FACTORY, 1289
 - DDS_ENTITYFACTORY_QOS_POLICY_NAME, 1290
- entity_factory
 - DDS_DomainParticipantFactoryQos, 1722
 - DDS_DomainParticipantQos, 1728
 - DDS_PublisherQos, 1919
 - DDS_SubscriberQos, 2011
- ENTITY_NAME, 1290
 - DDS_ENTITYNAME_QOS_POLICY_NAME, 1291
- enum_as_int
 - DDS_PrintFormatProperty, 1896
- evaluate
 - DDS_ContentFilter, 1574
- EVENT, 1291
 - DDS_EVENT_QOS_POLICY_NAME, 1292
- event
 - DDS_DomainParticipantQos, 1729
- event_settings
 - DDS_MonitoringDistributionSettings, 1839
- Exception Codes, 1219
 - DDS_BAD_MEMBER_ID_USER_EXCEPTION_CODE, 1220
 - DDS_BAD_MEMBER_NAME_USER_EXCEPTION_CODE, 1220
 - DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE, 1220
 - DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE, 1220
 - DDS_BADKIND_USER_EXCEPTION_CODE, 1220
 - DDS_BOUNDS_USER_EXCEPTION_CODE, 1220
 - DDS_ExceptionCode_t, 1219
 - DDS_IMMUTABLE_TYPECODE_SYSTEM_EXCEPTION_CODE, 1220
 - DDS_NO_EXCEPTION_CODE, 1219
 - DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE, 1220
 - DDS_SYSTEM_EXCEPTION_CODE, 1219
 - DDS_USER_EXCEPTION_CODE, 1219
- expects_inline_qos
 - DDS_DataReaderProtocolQosPolicy, 1597
- expiration_code
 - DDS_InvalidLocalAdvanceNoticeStatus, 1811
- expiration_message
 - DDS_InvalidLocalAdvanceNoticeStatus, 1811
- expiration_time
 - DDS_InvalidLocalAdvanceNoticeStatus, 1811
- expired_dropped_sample_count
 - DDS_DataReaderCacheStatus, 1589
- expression_parameters
 - DDS_ContentFilterProperty_t, 1580
- Extended Qos Support, 1350
- facility
 - NDDS_Config_LogMessage, 2119
- factory_data
 - DDS_ThreadFactory, 2028
- fast_heartbeat_period
 - DDS_RtpsReliableWriterProtocol_t, 1963
- file_name
 - DDS_PersistentStorageSettings, 1885
- Filter Use Cases, 995
- filter_class_name
 - DDS_ContentFilterProperty_t, 1579
 - DDS_TopicQuerySelection, 2050
- filter_data
 - DDS_ContentFilter, 1578
- filter_expression
 - DDS_ChannelSettings_t, 1564
 - DDS_ContentFilterProperty_t, 1580
 - DDS_LocatorFilter_t, 1828
 - DDS_TopicQuerySelection, 2050
- filter_name
 - DDS_LocatorFilterQosPolicy, 1829
 - DDS_MultiChannelQosPolicy, 1858
- filter_parameters
 - DDS_TopicQuerySelection, 2050
- filtered_sample_bytes
 - DDS_DataReaderProtocolStatus, 1605
 - DDS_DataWriterProtocolStatus, 1658
- filtered_sample_bytes_change
 - DDS_DataReaderProtocolStatus, 1605
 - DDS_DataWriterProtocolStatus, 1658
- filtered_sample_count
 - DDS_DataReaderProtocolStatus, 1604
 - DDS_DataWriterProtocolStatus, 1657
- filtered_sample_count_change
 - DDS_DataReaderProtocolStatus, 1605
 - DDS_DataWriterProtocolStatus, 1658
- finalize
 - DDS_ContentFilter, 1576
- first_available_sample_sequence_number
 - DDS_DataReaderProtocolStatus, 1609
 - DDS_DataWriterProtocolStatus, 1663
- first_available_sample_virtual_sequence_number
 - DDS_DataWriterProtocolStatus, 1663
- first_unacknowledged_sample_sequence_number
 - DDS_DataWriterProtocolStatus, 1663
- first_unacknowledged_sample_subscription_handle
 - DDS_DataWriterProtocolStatus, 1664
- first_unacknowledged_sample_virtual_sequence_number
 - DDS_DataWriterProtocolStatus, 1664
- first_unelapsed_keep_duration_sample_sequence_number
 - DDS_DataWriterProtocolStatus, 1664
- flag
 - DDS_SampleInfo, 1994
 - DDS_WriteParams_t, 2104
- FlatData Topic-Types, 209

- Flow Control, 863
- Flow Controllers, 556
 - DDS_DEFAULT_FLOW_CONTROLLER_NAME, 564
 - DDS_EDF_FLOW_CONTROLLER_SCHED_POLICY, 560
 - DDS_FIXED_RATE_FLOW_CONTROLLER_NAME, 564
 - DDS_FlowController, 559
 - DDS_FlowController_get_name, 561
 - DDS_FlowController_get_participant, 561
 - DDS_FlowController_get_property, 563
 - DDS_FlowController_set_property, 562
 - DDS_FlowController_trigger_flow, 563
 - DDS_FlowControllerSchedulingPolicy, 559
 - DDS_FlowControllerTokenBucketProperty_t_INITIALIZED, 558
 - DDS_HPF_FLOW_CONTROLLER_SCHED_POLICY, 561
 - DDS_ON_DEMAND_FLOW_CONTROLLER_NAME, 565
 - DDS_RR_FLOW_CONTROLLER_SCHED_POLICY, 560
- flow_controller_allocation
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1740
- flow_controller_hash_buckets
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1745
- flow_controller_name
 - DDS_PublishModeQosPolicy, 1922
- FlowController Use Cases, 978
- Foo, 2109
- FooBarReplier, 2110
 - Replier, 766
- FooBarReplier_create
 - Replier, 769
- FooBarReplier_create_w_params
 - Replier, 769
- FooBarReplier_get_reply_datawriter
 - Replier, 773
- FooBarReplier_get_request_datareader
 - Replier, 772
- FooBarReplier_read_request
 - Replier, 771
- FooBarReplier_read_requests
 - Replier, 771
- FooBarReplier_receive_request
 - Replier, 770
- FooBarReplier_receive_requests
 - Replier, 770
- FooBarReplier_return_loan
 - Replier, 773
- FooBarReplier_send_reply
 - Replier, 772
- FooBarReplier_take_request
 - Replier, 770
- FooBarReplier_take_requests
 - Replier, 771
- FooBarRequester, 2111
 - Requester, 748
- FooBarRequester_create
 - Requester, 752
- FooBarRequester_create_w_params
 - Requester, 752
- FooBarRequester_get_reply_datareader
 - Requester, 760
- FooBarRequester_get_request_datawriter
 - Requester, 759
- FooBarRequester_read_replies
 - Requester, 758
- FooBarRequester_read_replies_for_related_request
 - Requester, 759
- FooBarRequester_read_reply
 - Requester, 758
- FooBarRequester_read_reply_for_related_request
 - Requester, 759
- FooBarRequester_receive_replies
 - Requester, 754
- FooBarRequester_receive_reply
 - Requester, 754
- FooBarRequester_return_loan
 - Requester, 760
- FooBarRequester_send_request
 - Requester, 752
- FooBarRequester_send_request_w_params
 - Requester, 753
- FooBarRequester_take_replies
 - Requester, 756
- FooBarRequester_take_replies_for_related_request
 - Requester, 757
- FooBarRequester_take_reply
 - Requester, 755
- FooBarRequester_take_reply_for_related_request
 - Requester, 757
- FooBarSimpleReplier, 2112
 - Replier, 767
- FooBarSimpleReplier_create
 - Replier, 773
- FooBarSimpleReplier_create_w_params
 - Replier, 774
- FooBarSimpleReplier_delete
 - Replier, 774
- FooDataReader, 2112
- FooDataReader_as_datareader
 - DataReaders, 624
- FooDataReader_get_key_value
 - DataReaders, 647
- FooDataReader_is_data_consistent

- DataReaders, 649
- FooDataReader_lookup_instance
 - DataReaders, 648
- FooDataReader_narrow
 - DataReaders, 624
- FooDataReader_read
 - DataReaders, 625
- FooDataReader_read_instance
 - DataReaders, 635
- FooDataReader_read_instance_w_condition
 - DataReaders, 638
- FooDataReader_read_next_instance
 - DataReaders, 640
- FooDataReader_read_next_instance_w_condition
 - DataReaders, 643
- FooDataReader_read_next_sample
 - DataReaders, 633
- FooDataReader_read_w_condition
 - DataReaders, 630
- FooDataReader_return_loan
 - DataReaders, 646
- FooDataReader_take
 - DataReaders, 626
- FooDataReader_take_instance
 - DataReaders, 636
- FooDataReader_take_instance_w_condition
 - DataReaders, 639
- FooDataReader_take_next_instance
 - DataReaders, 642
- FooDataReader_take_next_instance_w_condition
 - DataReaders, 645
- FooDataReader_take_next_sample
 - DataReaders, 634
- FooDataReader_take_w_condition
 - DataReaders, 632
- FooDataWriter, 2113
- FooDataWriter_as_datawriter
 - Data Writers, 490
- FooDataWriter_create_data
 - Data Writers, 506
- FooDataWriter_create_data_w_params
 - Data Writers, 507
- FooDataWriter_delete_data
 - Data Writers, 507
- FooDataWriter_delete_data_w_params
 - Data Writers, 508
- FooDataWriter_discard_loan
 - Data Writers, 510
- FooDataWriter_dispose
 - Data Writers, 502
- FooDataWriter_dispose_w_params
 - Data Writers, 504
- FooDataWriter_dispose_w_timestamp
 - Data Writers, 503
- FooDataWriter_get_key_value
 - Data Writers, 505
- FooDataWriter_get_loan
 - Data Writers, 508
- FooDataWriter_lookup_instance
 - Data Writers, 506
- FooDataWriter_narrow
 - Data Writers, 490
- FooDataWriter_register_instance
 - Data Writers, 491
- FooDataWriter_register_instance_w_params
 - Data Writers, 493
- FooDataWriter_register_instance_w_timestamp
 - Data Writers, 492
- FooDataWriter_unregister_instance
 - Data Writers, 493
- FooDataWriter_unregister_instance_w_params
 - Data Writers, 496
- FooDataWriter_unregister_instance_w_timestamp
 - Data Writers, 495
- FooDataWriter_write
 - Data Writers, 496
- FooDataWriter_write_w_params
 - Data Writers, 501
- FooDataWriter_write_w_timestamp
 - Data Writers, 500
- FooSeq, 2113
- FooSeq_copy
 - Sequence Support, 1519
- FooSeq_copy_no_alloc
 - Sequence Support, 1518
- FooSeq_ensure_length
 - Sequence Support, 1516
- FooSeq_finalize
 - Sequence Support, 1525
- FooSeq_from_array
 - Sequence Support, 1519
- FooSeq_get
 - Sequence Support, 1517
- FooSeq_get_contiguous_buffer
 - Sequence Support, 1523
- FooSeq_get_discontiguous_buffer
 - Sequence Support, 1524
- FooSeq_get_length
 - Sequence Support, 1515
- FooSeq_get_maximum
 - Sequence Support, 1514
- FooSeq_get_reference
 - Sequence Support, 1517
- FooSeq_has_ownership
 - Sequence Support, 1524
- FooSeq_initialize
 - Sequence Support, 1513
- FooSeq_loan_contiguous

- Sequence Support, 1521
- FooSeq_loan_discontiguous
 - Sequence Support, 1522
- FooSeq_set_length
 - Sequence Support, 1515
- FooSeq_set_maximum
 - Sequence Support, 1514
- FooSeq_to_array
 - Sequence Support, 1520
- FooSeq_unloan
 - Sequence Support, 1522
- FooTypeSupport, 2114
- FooTypeSupport_copy_data
 - User Data Type Support, 216
- FooTypeSupport_create_data
 - User Data Type Support, 215
- FooTypeSupport_create_data_ex
 - User Data Type Support, 215
- FooTypeSupport_create_data_w_params
 - User Data Type Support, 216
- FooTypeSupport_data_to_string
 - User Data Type Support, 225
- FooTypeSupport_delete_data
 - User Data Type Support, 217
- FooTypeSupport_delete_data_ex
 - User Data Type Support, 217
- FooTypeSupport_delete_data_w_params
 - User Data Type Support, 218
- FooTypeSupport_deserialize_data_from_cdr_buffer
 - User Data Type Support, 225
- FooTypeSupport_finalize_data
 - User Data Type Support, 220
- FooTypeSupport_finalize_data_ex
 - User Data Type Support, 220
- FooTypeSupport_get_type_name
 - User Data Type Support, 221
- FooTypeSupport_get_typecode
 - User Data Type Support, 226
- FooTypeSupport_initialize_data
 - User Data Type Support, 218
- FooTypeSupport_initialize_data_ex
 - User Data Type Support, 219
- FooTypeSupport_print_data
 - User Data Type Support, 223
- FooTypeSupport_register_type
 - User Data Type Support, 221
- FooTypeSupport_serialize_data_to_cdr_buffer
 - User Data Type Support, 223
- FooTypeSupport_serialize_data_to_cdr_buffer_ex
 - User Data Type Support, 224
- FooTypeSupport_unregister_type
 - User Data Type Support, 222
- force_interface_poll_detection
 - NDDS_Transport_UDPv4_Property_t, 2141

- NDDS_Transport_UDPv4_WAN_Property_t, 2150
- NDDS_Transport_UDPv6_Property_t, 2161
- force_type_validation
 - DDS_TypeConsistencyEnforcementQosPolicy, 2079
- Fragmentation, 866
- frame_queue_size
 - NDDS_Utility_NetworkCaptureParams_t, 2167
- full_reliable_writer_cache
 - DDS_ReliableWriterCacheChangedStatus, 1944
- function_name
 - DDS_TransportMulticastMappingFunction_t, 2056
- Functional Categories, 833
- gather_send_buffer_count_max
 - NDDS_Transport_Property_t, 2124
- General Utilities and Compliance Configuration, 744
- generation_rank
 - DDS_SampleInfo, 1991
- group_coherent_set_sequence_number
 - DDS_CoherentSetInfo_t, 1567
- GROUP_DATA, 1293
 - DDS_GROUPDATA_QOS_POLICY_NAME, 1294
- group_data
 - DDS_PublicationBuiltinTopicData, 1910
 - DDS_PublisherQos, 1919
 - DDS_SubscriberQos, 2011
 - DDS_SubscriptionBuiltinTopicData, 2017
- group_guid
 - DDS_CoherentSetInfo_t, 1567
- GUID Support, 1210
 - DDS_GUID_AUTO, 1213
 - DDS_GUID_compare, 1212
 - DDS_GUID_copy, 1213
 - DDS_GUID_equals, 1212
 - DDS_GUID_t, 1211
 - DDS_GUID_UNKNOWN, 1213
 - DDS_GUID_ZERO, 1213
 - DDS_RTPS_EntityId_t, 1211
 - DDS_RTPS_GUID_t, 1211
 - DDS_RTPS_GuidPrefix_t, 1211
- handle
 - DDS_WriteParams_t, 2103
- handler_data
 - DDS_ConditionHandler, 1571
 - DDS_SampleHandler, 1984
- Heap Monitoring, 1477
 - NDDS_Utility_disable_heap_monitoring, 1480
 - NDDS_Utility_enable_heap_monitoring, 1479
 - NDDS_Utility_enable_heap_monitoring_w_params, 1480
 - NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT, 1479
 - NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT, 1479

- NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT_TOPIC, 1479
- NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT_TOPIC, 1479
- NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT_TOPIC, 1479
- NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_OUTPUT_FORMAT, 1479
- NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_OUTPUT_FORMAT, 1479
- NDDS_UTILITY_HEAP_MONITORING_PARAMS_INITIALIZER, 1478
- NDDS_UTILITY_HEAP_MONITORING_PARAMS_t, 1478
- NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_FORMAT, 1479
- NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_OUTPUT_FORMAT, 1478
- NDDS_UTILITY_PAUSE_HEAP_MONITORING, 1480
- NDDS_UTILITY_RESUME_HEAP_MONITORING, 1481
- NDDS_UTILITY_TAKE_HEAP_SNAPSHOT, 1481
- Heap Support in C, 1392
 - DDS_Heap_calloc, 1393
 - DDS_Heap_free, 1393
 - DDS_Heap_malloc, 1393
- heartbeat_period
 - DDS_RtpsReliableWriterProtocol_t, 1963
- heartbeat_suppression_duration
 - DDS_RtpsReliableReaderProtocol_t, 1956
- heartbeats_per_max_samples
 - DDS_RtpsReliableWriterProtocol_t, 1967
- high
 - DDS_SequenceNumber_t, 1999
- high_watermark
 - DDS_RtpsReliableWriterProtocol_t, 1962
- high_watermark_reliable_writer_cache
 - DDS_ReliableWriterCacheChangedStatus, 1944
- HISTORY, 1292
 - DDS_HISTORY_QOS_POLICY_NAME, 1293
 - DDS_HistoryQosPolicyKind, 1292
 - DDS_KEEP_ALL_HISTORY_QOS, 1293
 - DDS_KEEP_LAST_HISTORY_QOS, 1293
- history
 - DDS_DataReaderQos, 1614
 - DDS_DataWriterQos, 1669
 - DDS_TopicBuiltinTopicData, 2038
 - DDS_TopicQos, 2045
- history_depth
 - DDS_DurabilityServiceQosPolicy, 1765
- history_kind
 - DDS_DurabilityServiceQosPolicy, 1765
- host_port
 - NDDS_Transport_UDP_WAN_CommPortsMappingInfo, 2133
- howto_prototypes.h, 2183
- DDS_StructMember, 2008
- DDS_StructMember, 2090
- identity
- DDS_MinimalParams_t, 2102
- ignore_default_domain_announcements
- DDS_OrbWriterCompressPolicy, 1705
- ignore_enum_literal_names
- DDS_TypeConsistencyEnforcementQosPolicy, 2079
- ignore_environment_profile
- DDS_ProfileQosPolicy, 1899
- ignore_loopback_interface
- NDDS_Transport_UDPv4_Property_t, 2137
- NDDS_Transport_UDPv4_WAN_Property_t, 2146
- NDDS_Transport_UDPv6_Property_t, 2157
- ignore_member_names
- DDS_TypeConsistencyEnforcementQosPolicy, 2078
- ignore_nonrunning_interfaces
- NDDS_Transport_UDPv4_Property_t, 2138
- NDDS_Transport_UDPv4_WAN_Property_t, 2147
- NDDS_Transport_UDPv6_Property_t, 2158
- ignore_nonup_interfaces
- NDDS_Transport_UDPv4_Property_t, 2138
- NDDS_Transport_UDPv4_WAN_Property_t, 2146
- ignore_resource_profile
- DDS_ProfileQosPolicy, 1900
- ignore_sequence_bounds
- DDS_TypeConsistencyEnforcementQosPolicy, 2077
- ignore_string_bounds
- DDS_TypeConsistencyEnforcementQosPolicy, 2078
- ignore_user_profile
- DDS_ProfileQosPolicy, 1899
- ignored_entity_allocation
- DDS_DomainParticipantResourceLimitsQosPolicy, 1738
- ignored_entity_hash_buckets
- DDS_DomainParticipantResourceLimitsQosPolicy, 1744
- ignored_entity_replacement_kind
- DDS_DomainParticipantResourceLimitsQosPolicy, 1755
- Immutable QoS, 914
- inactivate_nonprogressing_readers
- DDS_RtpsReliableWriterProtocol_t, 1966
- inactive_count
- DDS_ReliableReaderActivityChangedStatus, 1941
- inactive_count_change
- DDS_ReliableReaderActivityChangedStatus, 1942
- include_root_elements
- DDS_PrintFormatProperty, 1896
- incomplete_coherent_set
- DDS_CoherentSetInfo_t, 1568
- incremental_count
- DDS_AllocationSettings_t, 1538

- indent
 - DDS_QosPrintFormat, 1927
 - DDS_TypeCodePrintFormatProperty, 2073
- Infrastructure Module, 716
- initial_active_topic_queries
 - DDS_DataWriterResourceLimitsQosPolicy, 1680
- initial_batches
 - DDS_DataWriterResourceLimitsQosPolicy, 1676
- initial_concurrent_blocking_threads
 - DDS_DataWriterResourceLimitsQosPolicy, 1675
- initial_count
 - DDS_AllocationSettings_t, 1538
 - DDS_EventQosPolicy, 1794
- initial_fragmented_samples
 - DDS_BuiltinTopicReaderResourceLimits_t, 1561
 - DDS_DataReaderResourceLimitsQosPolicy, 1627
- initial_infos
 - DDS_BuiltinTopicReaderResourceLimits_t, 1559
 - DDS_DataReaderResourceLimitsQosPolicy, 1624
- initial_instances
 - DDS_ResourceLimitsQosPolicy, 1952
- initial_objects_per_thread
 - DDS_SystemResourceLimitsQosPolicy, 2025
- initial_outstanding_reads
 - DDS_BuiltinTopicReaderResourceLimits_t, 1559
 - DDS_DataReaderResourceLimitsQosPolicy, 1625
- initial_participant_announcements
 - DDS_DiscoveryConfigQosPolicy, 1694
- initial_peers
 - DDS_DiscoveryQosPolicy, 1716
- initial_records
 - DDS_DatabaseQosPolicy, 1584
- initial_remote_virtual_writers
 - DDS_DataReaderResourceLimitsQosPolicy, 1630
- initial_remote_virtual_writers_per_instance
 - DDS_DataReaderResourceLimitsQosPolicy, 1631
- initial_remote_writers
 - DDS_DataReaderResourceLimitsQosPolicy, 1623
- initial_remote_writers_per_instance
 - DDS_DataReaderResourceLimitsQosPolicy, 1624
- initial_samples
 - DDS_BuiltinTopicReaderResourceLimits_t, 1558
 - DDS_ResourceLimitsQosPolicy, 1952
- initial_topic_queries
 - DDS_DataReaderResourceLimitsQosPolicy, 1634
- initial_virtual_sequence_number
 - DDS_DataWriterProtocolQosPolicy, 1653
- initial_virtual_writers
 - DDS_DataWriterResourceLimitsQosPolicy, 1679
- initial_weak_references
 - DDS_DatabaseQosPolicy, 1585
- initialize_writer_loaned_sample
 - DDS_DataWriterResourceLimitsQosPolicy, 1681
- Installing Transport Plugins, 725
- NDDS_Transport_create_plugin, 729
- NDDS_Transport_Handle_is_nil, 730
- NDDS_TRANSPORT_HANDLE_NIL, 736
- NDDS_Transport_Handle_t, 729
- NDDS_Transport_Support_add_receive_route, 733
- NDDS_Transport_Support_add_send_route, 732
- NDDS_Transport_Support_get_builtin_transport_property, 734
- NDDS_Transport_Support_get_transport_plugin, 735
- NDDS_Transport_Support_lookup_transport, 732
- NDDS_Transport_Support_register_transport, 730
- NDDS_Transport_Support_set_builtin_transport_property, 735
- Instance Lifecycle, 867
- Instance States, 713
 - DDS_ALIVE_INSTANCE_STATE, 715
 - DDS_ANY_INSTANCE_STATE, 715
 - DDS_InstanceStateKind, 714
 - DDS_InstanceStateMask, 714
 - DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE, 715
 - DDS_NOT_ALIVE_INSTANCE_STATE, 715
 - DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE, 715
- instance_handle
 - DDS_SampleInfo, 1990
- instance_hash_buckets
 - DDS_ResourceLimitsQosPolicy, 1953
- instance_id
 - DDS_ServiceRequest, 2001
- instance_replacement
 - DDS_DataReaderResourceLimitsQosPolicy, 1635
 - DDS_DataWriterResourceLimitsQosPolicy, 1677
- instance_state
 - DDS_SampleInfo, 1989
- instance_state_consistency_kind
 - DDS_ReliabilityQosPolicy, 1940
- instance_states
 - DDS_ReadConditionParams, 1930
- interceptor
 - DDS_EndpointTrustAlgorithmInfo, 1786
 - DDS_ParticipantTrustAlgorithmInfo, 1877
- Interface, 1019
 - NDDS_TRANSPORT_INTERFACE_OFF, 1020
 - NDDS_TRANSPORT_INTERFACE_ON, 1020
 - NDDS_Transport_Interface_Status_t, 1019
- interface_poll_period
 - NDDS_Transport_UDPv4_Property_t, 2141
 - NDDS_Transport_UDPv4_WAN_Property_t, 2150
 - NDDS_Transport_UDPv6_Property_t, 2161
- is_key
 - DDS_StructMember, 2008
 - DDS_ValueMember, 2089

- is_license_message
 - NDDS_Config_LogMessage, 2118
- is_optional
 - DDS_StructMember, 2008
 - DDS_ValueMember, 2090
- is_pointer
 - DDS_StructMember, 2008
 - DDS_UnionMember, 2084
 - DDS_ValueMember, 2089
- is_security_message
 - NDDS_Config_LogMessage, 2118
- is_standalone
 - DDS_QosPrintFormat, 1926
- join_multicast_group_timeout
 - NDDS_Transport_UDPv4_Property_t, 2142
 - NDDS_Transport_UDPv6_Property_t, 2162
- journal_kind
 - DDS_PersistentStorageSettings, 1886
- keep_minimum_state_for_instances
 - DDS_DataReaderResourceLimitsQosPolicy, 1633
- key
 - DDS_ParticipantBuiltinTopicData, 1871
 - DDS_PublicationBuiltinTopicData, 1907
 - DDS_SubscriptionBuiltinTopicData, 2014
 - DDS_TopicBuiltinTopicData, 2036
 - KeyedOctets Built-in Type, 1199
 - KeyedString Built-in Type, 1150
- key_establishment
 - DDS_ParticipantTrustAlgorithmInfo, 1877
- key_only_filter
 - DDS_ExpressionProperty, 1795
- KeyedOctets Built-in Type, 1169
 - DDS_KeyedOctets, 1174
 - DDS_KeyedOctets_delete, 1176
 - DDS_KeyedOctets_new, 1175
 - DDS_KeyedOctets_new_w_size, 1176
 - DDS_KeyedOctetsDataReader, 1175
 - DDS_KeyedOctetsDataReader_as_datareader, 1192
 - DDS_KeyedOctetsDataReader_get_key_value, 1197
 - DDS_KeyedOctetsDataReader_get_key_value_w_key, 1198
 - DDS_KeyedOctetsDataReader_lookup_instance, 1198
 - DDS_KeyedOctetsDataReader_lookup_instance_w_key, 1198
 - DDS_KeyedOctetsDataReader_narrow, 1192
 - DDS_KeyedOctetsDataReader_read, 1192
 - DDS_KeyedOctetsDataReader_read_instance, 1194
 - DDS_KeyedOctetsDataReader_read_instance_w_condition, 1195
 - DDS_KeyedOctetsDataReader_read_next_instance, 1196
 - DDS_KeyedOctetsDataReader_read_next_instance_w_condition, 1196
 - DDS_KeyedOctetsDataReader_read_next_sample, 1194
 - DDS_KeyedOctetsDataReader_read_w_condition, 1193
 - DDS_KeyedOctetsDataReader_return_loan, 1197
 - DDS_KeyedOctetsDataReader_take, 1193
 - DDS_KeyedOctetsDataReader_take_instance, 1195
 - DDS_KeyedOctetsDataReader_take_instance_w_condition, 1195
 - DDS_KeyedOctetsDataReader_take_next_instance, 1196
 - DDS_KeyedOctetsDataReader_take_next_instance_w_condition, 1197
 - DDS_KeyedOctetsDataReader_take_next_sample, 1194
 - DDS_KeyedOctetsDataReader_take_w_condition, 1193
 - DDS_KeyedOctetsDataWriter, 1175
 - DDS_KeyedOctetsDataWriter_as_datawriter, 1181
 - DDS_KeyedOctetsDataWriter_create_data, 1184
 - DDS_KeyedOctetsDataWriter_delete_data, 1184
 - DDS_KeyedOctetsDataWriter_dispose, 1189
 - DDS_KeyedOctetsDataWriter_dispose_w_key, 1189
 - DDS_KeyedOctetsDataWriter_dispose_w_key_w_timestamp, 1190
 - DDS_KeyedOctetsDataWriter_dispose_w_timestamp, 1190
 - DDS_KeyedOctetsDataWriter_get_key_value, 1190
 - DDS_KeyedOctetsDataWriter_get_key_value_w_key, 1191
 - DDS_KeyedOctetsDataWriter_lookup_instance, 1191
 - DDS_KeyedOctetsDataWriter_lookup_instance_w_key, 1191
 - DDS_KeyedOctetsDataWriter_narrow, 1181
 - DDS_KeyedOctetsDataWriter_register_instance, 1181
 - DDS_KeyedOctetsDataWriter_register_instance_w_key, 1182
 - DDS_KeyedOctetsDataWriter_register_instance_w_key_w_timestamp, 1182
 - DDS_KeyedOctetsDataWriter_register_instance_w_timestamp, 1182
 - DDS_KeyedOctetsDataWriter_unregister_instance, 1183
 - DDS_KeyedOctetsDataWriter_unregister_instance_w_key, 1183
 - DDS_KeyedOctetsDataWriter_unregister_instance_w_key_w_timestamp, 1184
 - DDS_KeyedOctetsDataWriter_unregister_instance_w_timestamp, 1183
 - DDS_KeyedOctetsDataWriter_write, 1185

- DDS_KeyedOctetsDataWriter_write_octets_seq_w_key, 1147
- 1186 DDS_KeyedStringDataReader_read_next_instance_w_condition, 1148
- DDS_KeyedOctetsDataWriter_write_octets_seq_w_key_w_params, 1189
- 1189 DDS_KeyedStringDataReader_read_next_sample, 1145
- DDS_KeyedOctetsDataWriter_write_octets_seq_w_key_w_timestamp, 1187
- 1187 DDS_KeyedStringDataReader_read_w_condition, 1145
- DDS_KeyedOctetsDataWriter_write_octets_w_key, 1185
- 1185 DDS_KeyedStringDataReader_return_loan, 1149
- DDS_KeyedOctetsDataWriter_write_octets_w_key_w_params, 1188
- 1188 DDS_KeyedStringDataReader_take, 1144
- DDS_KeyedOctetsDataWriter_write_octets_w_key_w_timestamp, 1186
- 1186 DDS_KeyedStringDataReader_take_instance, 1146
- 1186 DDS_KeyedStringDataReader_take_instance_w_condition, 1147
- DDS_KeyedOctetsDataWriter_write_w_params, 1188
- 1188 DDS_KeyedStringDataReader_take_next_instance, 1148
- DDS_KeyedOctetsDataWriter_write_w_timestamp, 1186
- 1186 DDS_KeyedStringDataReader_take_next_instance_w_condition, 1148
- DDS_KeyedOctetsTypeSupport_data_to_string, 1180
- 1180 DDS_KeyedStringDataReader_take_next_sample, 1146
- DDS_KeyedOctetsTypeSupport_deserialize_data_from_cdr_buffer, 1180
- 1180 DDS_KeyedStringDataReader_take_w_condition, 1145
- DDS_KeyedOctetsTypeSupport_get_type_name, 1178
- 1178 DDS_KeyedStringDataWriter, 1129
- DDS_KeyedOctetsTypeSupport_get_typecode, 1179
- 1179 DDS_KeyedStringDataWriter_as_datawriter, 1135
- DDS_KeyedOctetsTypeSupport_print_data, 1179
- 1179 DDS_KeyedStringDataWriter_create_data, 1138
- DDS_KeyedOctetsTypeSupport_register_type, 1176
- 1176 DDS_KeyedStringDataWriter_delete_data, 1138
- DDS_KeyedOctetsTypeSupport_serialize_data_to_cdr_buffer, 1179
- 1179 DDS_KeyedStringDataWriter_dispose, 1141
- DDS_KeyedOctetsTypeSupport_serialize_data_to_cdr_buffer_ex, 1180
- 1180 DDS_KeyedStringDataWriter_dispose_w_key, 1141
- DDS_KeyedOctetsTypeSupport_unregister_type, 1177
- 1177 DDS_KeyedStringDataWriter_dispose_w_key_w_timestamp, 1142
- key, 1199
- length, 1199
- value, 1199
- KeyedString Built-in Type, 1123
- DDS_KeyedString, 1129
- DDS_KeyedString_delete, 1130
- DDS_KeyedString_new, 1129
- DDS_KeyedString_new_w_size, 1130
- DDS_KeyedStringDataReader, 1129
- DDS_KeyedStringDataReader_as_datareader, 1144
- DDS_KeyedStringDataReader_get_key_value, 1149
- DDS_KeyedStringDataReader_get_key_value_w_key, 1149
- DDS_KeyedStringDataReader_lookup_instance, 1150
- DDS_KeyedStringDataReader_lookup_instance_w_key, 1150
- DDS_KeyedStringDataReader_narrow, 1143
- DDS_KeyedStringDataReader_read, 1144
- DDS_KeyedStringDataReader_read_instance, 1146
- DDS_KeyedStringDataReader_read_instance_w_condition, 1147
- DDS_KeyedStringDataReader_read_next_instance, 1147
- DDS_KeyedStringDataWriter, 1129
- DDS_KeyedStringDataWriter_as_datawriter, 1135
- DDS_KeyedStringDataWriter_create_data, 1138
- DDS_KeyedStringDataWriter_delete_data, 1138
- DDS_KeyedStringDataWriter_dispose, 1141
- DDS_KeyedStringDataWriter_dispose_w_key, 1141
- DDS_KeyedStringDataWriter_dispose_w_key_w_timestamp, 1142
- DDS_KeyedStringDataWriter_dispose_w_timestamp, 1141
- DDS_KeyedStringDataWriter_get_key_value, 1142
- DDS_KeyedStringDataWriter_get_key_value_w_key, 1142
- DDS_KeyedStringDataWriter_lookup_instance, 1143
- DDS_KeyedStringDataWriter_lookup_instance_w_key, 1143
- DDS_KeyedStringDataWriter_narrow, 1135
- DDS_KeyedStringDataWriter_register_instance, 1135
- DDS_KeyedStringDataWriter_register_instance_w_key, 1136
- DDS_KeyedStringDataWriter_register_instance_w_key_w_timestamp, 1136
- DDS_KeyedStringDataWriter_register_instance_w_timestamp, 1136
- DDS_KeyedStringDataWriter_unregister_instance, 1137
- DDS_KeyedStringDataWriter_unregister_instance_w_key, 1137
- DDS_KeyedStringDataWriter_unregister_instance_w_key_w_timestamp, 1138
- DDS_KeyedStringDataWriter_unregister_instance_w_timestamp, 1137
- DDS_KeyedStringDataWriter_write, 1139

- DDS_KeyedStringDataWriter_write_string_w_key, 1139
- DDS_KeyedStringDataWriter_write_string_w_key_w_params, 1140
- DDS_KeyedStringDataWriter_write_string_w_key_w_timestamp, 1140
- DDS_KeyedStringDataWriter_write_w_params, 1140
- DDS_KeyedStringDataWriter_write_w_timestamp, 1139
- DDS_KeyedStringTypeSupport_data_to_string, 1134
- DDS_KeyedStringTypeSupport_deserialize_data_from_cdr_buffer, 1134
- DDS_KeyedStringTypeSupport_get_type_name, 1132
- DDS_KeyedStringTypeSupport_get_typecode, 1133
- DDS_KeyedStringTypeSupport_print_data, 1133
- DDS_KeyedStringTypeSupport_register_type, 1130
- DDS_KeyedStringTypeSupport_serialize_data_to_cdr_buffer, 1133
- DDS_KeyedStringTypeSupport_serialize_data_to_cdr_buffer, 1134
- DDS_KeyedStringTypeSupport_unregister_type, 1131
- key, 1150
- value, 1151
- kind
 - DDS_DestinationOrderQosPolicy, 1688
 - DDS_DurabilityQosPolicy, 1762
 - DDS_HistoryQosPolicy, 1807
 - DDS_LivelinessQosPolicy, 1825
 - DDS_Locator_t, 1827
 - DDS_OwnershipQosPolicy, 1868
 - DDS_PrintFormatProperty, 1895
 - DDS_PublishModeQosPolicy, 1922
 - DDS_ReliabilityQosPolicy, 1939
 - DDS_ServiceQosPolicy, 2000
 - DDS_TopicQuerySelection, 2051
 - DDS_TransportMulticastQosPolicy, 2059
 - DDS_TypeConsistencyEnforcementQosPolicy, 2077
- labels
 - DDS_UnionMember, 2084
- Large Data Use Cases, 1001
- last_available_sample_sequence_number
 - DDS_DataReaderProtocolStatus, 1609
 - DDS_DataWriterProtocolStatus, 1663
- last_available_sample_virtual_sequence_number
 - DDS_DataWriterProtocolStatus, 1663
- last_committed_sample_sequence_number
 - DDS_DataReaderProtocolStatus, 1609
- last_corrupted_message_timestamp
 - DDS_DomainParticipantProtocolStatus, 1726
- last_instance_handle
 - DDS_OfferedDeadlineMissedStatus, 1862
- DDS_ReliableReaderActivityChangedStatus, 1942
- DDS_RequestedDeadlineMissedStatus, 1947
- DDS_SampleRejectedStatus, 1998
- last_policy_id
- DDS_OfferedIncompatibleQosStatus, 1863
- DDS_RequestedIncompatibleQosStatus, 1948
- last_publication_handle
- DDS_LivelinessChangedStatus, 1821
- DDS_SubscriptionMatchedStatus, 2024
- last_reason
- DDS_SampleLostStatus, 1997
- DDS_SampleRejectedStatus, 1998
- last_request_handle
- DDS_ServiceRequestAcceptedStatus, 2003
- last_subscription_handle
- DDS_PublicationMatchedStatus, 1916
- late_joiner_heartbeat_period
- DDS_RtpsReliableWriterProtocol_t, 1964
- LATENCY_BUDGET, 1294
- DDS_LATENCYBUDGET_QOS_POLICY_NAME, 1295
- latency_budget
 - DDS_DataReaderQos, 1613
 - DDS_DataWriterQos, 1668
 - DDS_PublicationBuiltinTopicData, 1908
 - DDS_SubscriptionBuiltinTopicData, 2015
 - DDS_TopicBuiltinTopicData, 2037
 - DDS_TopicQos, 2044
- lease_duration
 - DDS_LivelinessQosPolicy, 1825
- length
 - KeyedOctets Built-in Type, 1199
 - Octets Built-in Type, 1168
- level
 - DDS_AsyncWaitSetProperty_t, 1547
 - NDDS_Config_LogMessage, 2118
- LIFESPAN, 1295
 - DDS_LIFESPAN_QOS_POLICY_NAME, 1295
- lifespan
 - DDS_DataWriterQos, 1669
 - DDS_PublicationBuiltinTopicData, 1908
 - DDS_TopicBuiltinTopicData, 2038
 - DDS_TopicQos, 2045
- listener
 - RTI_Connext_ReplierParams, 2172
- listener_data
 - DDS_AsyncWaitSetListener, 1544
 - DDS_Listener, 1819
- LIVELINESS, 1296
 - DDS_AUTOMATIC_LIVELINESS_QOS, 1297
 - DDS_LIVELINESS_QOS_POLICY_NAME, 1297
 - DDS_LivelinessQosPolicyKind, 1296
 - DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS, 1297

- DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS, 1297
- Liveliness, 874
- liveliness
 - DDS_DataReaderQos, 1613
 - DDS_DataWriterQos, 1668
 - DDS_PublicationBuiltinTopicData, 1908
 - DDS_SubscriptionBuiltinTopicData, 2015
 - DDS_TopicBuiltinTopicData, 2037
 - DDS_TopicQos, 2044
- local_publisher_allocation
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1736
- local_publisher_hash_buckets
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1741
- local_reader_allocation
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1735
- local_reader_hash_buckets
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1741
- local_subscriber_allocation
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1736
- local_subscriber_hash_buckets
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1742
- local_topic_allocation
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1736
- local_topic_hash_buckets
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1742
- local_writer_allocation
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1735
- local_writer_hash_buckets
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1741
- locator_filter
 - DDS_PublicationBuiltinTopicData, 1912
- locator_filters
 - DDS_LocatorFilterQosPolicy, 1829
- locator_reachability_assert_period
 - DDS_DiscoveryConfigQosPolicy, 1707
- locator_reachability_change_detection_period
 - DDS_DiscoveryConfigQosPolicy, 1708
- locator_reachability_lease_duration
 - DDS_DiscoveryConfigQosPolicy, 1708
- LOCATORFILTER, 1297
 - DDS_LOCATORFILTER_QOS_POLICY_NAME, 1298
- locators

- DDS_LocatorFilter_t, 1828
- LOGGING, 1298
- Logging, 875, 1438
 - NDDS_CONFIG_LOG_CATEGORY_ALL, 1446
 - NDDS_CONFIG_LOG_CATEGORY_API, 1445
 - NDDS_CONFIG_LOG_CATEGORY_COMMUNICATION, 1445
 - NDDS_CONFIG_LOG_CATEGORY_DATABASE, 1445
 - NDDS_CONFIG_LOG_CATEGORY_DISCOVERY, 1446
 - NDDS_CONFIG_LOG_CATEGORY_ENTITIES, 1445
 - NDDS_CONFIG_LOG_CATEGORY_PLATFORM, 1445
 - NDDS_CONFIG_LOG_CATEGORY_SECURITY, 1446
 - NDDS_CONFIG_LOG_CATEGORY_USER, 1446
 - NDDS_CONFIG_LOG_FACILITY_MIDDLEWARE, 1447
 - NDDS_CONFIG_LOG_FACILITY_SECURITY_EVENT, 1447
 - NDDS_CONFIG_LOG_FACILITY_SERVICE, 1447
 - NDDS_CONFIG_LOG_FACILITY_USER, 1447
 - NDDS_CONFIG_LOG_LEVEL_DEBUG, 1444
 - NDDS_CONFIG_LOG_LEVEL_ERROR, 1444
 - NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR, 1444
 - NDDS_CONFIG_LOG_LEVEL_STATUS_LOCAL, 1444
 - NDDS_CONFIG_LOG_LEVEL_STATUS_REMOTE, 1444
 - NDDS_CONFIG_LOG_LEVEL_WARNING, 1444
 - NDDS_CONFIG_LOG_PRINT_FORMAT_DEBUG, 1446
 - NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT, 1446
 - NDDS_CONFIG_LOG_PRINT_FORMAT_MAXIMAL, 1446
 - NDDS_CONFIG_LOG_PRINT_FORMAT_MINIMAL, 1446
 - NDDS_CONFIG_LOG_PRINT_FORMAT_TIMESTAMPED, 1446
 - NDDS_CONFIG_LOG_PRINT_FORMAT_VERBOSE, 1446
 - NDDS_CONFIG_LOG_PRINT_FORMAT_VERBOSE_TIMESTAMPED, 1446
 - NDDS_CONFIG_LOG_VERBOSITY_ERROR, 1443
 - NDDS_CONFIG_LOG_VERBOSITY_SILENT, 1443
 - NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL, 1443
 - NDDS_CONFIG_LOG_VERBOSITY_STATUS_LOCAL, 1443
 - NDDS_CONFIG_LOG_VERBOSITY_STATUS_REMOTE, 1443

- 1443
- NDDS_CONFIG_LOG_VERBOSITY_WARNING, 1443
- NDDS_Config_LogCategory, 1445
- NDDS_Config_LogFacility, 1446
- NDDS_Config_Logger_alert, 1453
- NDDS_Config_Logger_critical, 1453
- NDDS_Config_Logger_debug, 1456
- NDDS_Config_Logger_emergency, 1452
- NDDS_Config_Logger_error, 1454
- NDDS_Config_Logger_get_instance, 1448
- NDDS_Config_Logger_get_instancel, 1449
- NDDS_Config_Logger_get_output_device, 1456
- NDDS_Config_Logger_get_output_file, 1450
- NDDS_Config_Logger_get_print_format, 1451
- NDDS_Config_Logger_get_print_format_by_log_level, 1452
- NDDS_Config_Logger_get_verbosity, 1449
- NDDS_Config_Logger_get_verbosity_by_category, 1449
- NDDS_Config_Logger_informational, 1455
- NDDS_Config_Logger_notice, 1455
- NDDS_Config_Logger_set_output_device, 1457
- NDDS_Config_Logger_set_output_file, 1450
- NDDS_Config_Logger_set_output_file_name, 1451
- NDDS_Config_Logger_set_output_file_set, 1451
- NDDS_Config_Logger_set_print_format, 1452
- NDDS_Config_Logger_set_print_format_by_log_level, 1452
- NDDS_Config_Logger_set_verbosity, 1450
- NDDS_Config_Logger_set_verbosity_by_category, 1450
- NDDS_Config_Logger_warning, 1454
- NDDS_Config_LoggerDevice_INITIALIZER, 1442
- NDDS_Config_LoggerDeviceCloseFnc, 1443
- NDDS_Config_LoggerDeviceWriteFnc, 1442
- NDDS_Config_LogLevel, 1444
- NDDS_Config_LogPrintFormat, 1446
- NDDS_Config_LogVerbosity, 1443
- NDDS_CONFIG_SYSLOG_LEVEL_ALERT, 1444
- NDDS_CONFIG_SYSLOG_LEVEL_CRITICAL, 1445
- NDDS_CONFIG_SYSLOG_LEVEL_DEBUG, 1445
- NDDS_CONFIG_SYSLOG_LEVEL_EMERGENCY, 1444
- NDDS_CONFIG_SYSLOG_LEVEL_ERROR, 1445
- NDDS_CONFIG_SYSLOG_LEVEL_INFORMATIONAL, 1445
- NDDS_CONFIG_SYSLOG_LEVEL_NOTICE, 1445
- NDDS_CONFIG_SYSLOG_LEVEL_WARNING, 1445
- NDDS_CONFIG_SYSLOG_VERBOSITY_ALERT, 1448
- NDDS_CONFIG_SYSLOG_VERBOSITY_CRITICAL, 1448
- NDDS_CONFIG_SYSLOG_VERBOSITY_DEBUG, 1448
- NDDS_CONFIG_SYSLOG_VERBOSITY_EMERGENCY, 1447
- NDDS_CONFIG_SYSLOG_VERBOSITY_ERROR, 1448
- NDDS_CONFIG_SYSLOG_VERBOSITY_INFORMATIONAL, 1448
- NDDS_CONFIG_SYSLOG_VERBOSITY_NOTICE, 1448
- NDDS_CONFIG_SYSLOG_VERBOSITY_SILENT, 1447
- NDDS_CONFIG_SYSLOG_VERBOSITY_WARNING, 1448
- NDDS_Config_SyslogLevel, 1444
- NDDS_Config_SyslogVerbosity, 1447
- logging
 - DDS_DomainParticipantFactoryQos, 1722
- Logging and Version, 744
- logging_settings
 - DDS_MonitoringDistributionSettings, 1840
- logs
 - DDS_MonitoringTelemetryData, 1856
- low
 - DDS_SequenceNumber_t, 1999
- low_watermark
 - DDS_RtpsReliableWriterProtocol_t, 1961
- low_watermark_reliable_writer_cache
 - DDS_ReliableWriterCacheChangedStatus, 1944
- major
 - DDS_ProductVersion_t, 1897
 - DDS_ProtocolVersion_t, 1904
 - NDDS_Config_LibraryVersion_t, 2115
- mapping_function
 - DDS_TransportMulticastMapping_t, 2055
- mask
 - DDS_ThreadSettings_t, 2029
 - DDS_TransportBuiltinQosPolicy, 2052
- Matching, 963
- matching_reader_writer_pair_allocation
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1738
- matching_reader_writer_pair_hash_buckets
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1744
- matching_writer_reader_pair_allocation
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1738
- matching_writer_reader_pair_hash_buckets
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1743
- max_active_topic_queries
 - DDS_DataWriterResourceLimitsQosPolicy, 1680

- max_app_ack_remote_readers
 - DDS_DataWriterResourceLimitsQosPolicy, 1680
- max_app_ack_response_length
 - DDS_DataReaderResourceLimitsQosPolicy, 1633
- max_batches
 - DDS_DataWriterResourceLimitsQosPolicy, 1677
- max_blocking_time
 - DDS_ReliabilityQosPolicy, 1939
- max_bytes_per_file
 - DDS_LoggingQosPolicy, 1833
- max_bytes_per_nack_response
 - DDS_RtpsReliableWriterProtocol_t, 1969
- max_concurrent_blocking_threads
 - DDS_DataWriterResourceLimitsQosPolicy, 1675
- max_count
 - DDS_AllocationSettings_t, 1538
 - DDS_EventQosPolicy, 1794
- max_data_availability_waiting_time
 - DDS_AvailabilityQosPolicy, 1550
- max_data_bytes
 - DDS_BatchQosPolicy, 1553
- max_endpoint_availability_waiting_time
 - DDS_AvailabilityQosPolicy, 1550
- max_endpoint_group_cumulative_characters
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1755
- max_endpoint_groups
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1754
- max_event_count
 - DDS_WaitSetProperty_t, 2092
- max_event_delay
 - DDS_WaitSetProperty_t, 2092
- max_files
 - DDS_LoggingQosPolicy, 1834
- max_flush_delay
 - DDS_BatchQosPolicy, 1554
- max_fragmented_samples
 - DDS_BuiltinTopicReaderResourceLimits_t, 1561
 - DDS_DataReaderResourceLimitsQosPolicy, 1626
- max_fragmented_samples_per_remote_writer
 - DDS_BuiltinTopicReaderResourceLimits_t, 1562
 - DDS_DataReaderResourceLimitsQosPolicy, 1627
- max_fragments_per_sample
 - DDS_BuiltinTopicReaderResourceLimits_t, 1562
 - DDS_DataReaderResourceLimitsQosPolicy, 1628
- max_gather_destinations
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1745
- max_heartbeat_response_delay
 - DDS_RtpsReliableReaderProtocol_t, 1955
- max_heartbeat_retries
 - DDS_RtpsReliableWriterProtocol_t, 1966
- max_historical_logs
 - DDS_MonitoringLoggingDistributionSettings, 1845
- max_infos
 - DDS_BuiltinTopicReaderResourceLimits_t, 1559
 - DDS_DataReaderResourceLimitsQosPolicy, 1623
- max_initial_participant_announcement_period
 - DDS_DiscoveryConfigQosPolicy, 1695
- max_instances
 - DDS_DurabilityServiceQosPolicy, 1766
 - DDS_ResourceLimitsQosPolicy, 1951
- max_interface_count
 - NDDS_Transport_Property_t, 2128
- max_liveliness_loss_detection_period
 - DDS_DiscoveryConfigQosPolicy, 1694
- max_nack_response_delay
 - DDS_RtpsReliableWriterProtocol_t, 1968
- max_objects_per_thread
 - DDS_SystemResourceLimitsQosPolicy, 2025
- max_outstanding_reads
 - DDS_BuiltinTopicReaderResourceLimits_t, 1560
 - DDS_DataReaderResourceLimitsQosPolicy, 1625
- max_partition_cumulative_characters
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1748
- max_partitions
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1748
- max_query_condition_filters
 - DDS_DataReaderResourceLimitsQosPolicy, 1632
- max_remote_reader_filters
 - DDS_DataWriterResourceLimitsQosPolicy, 1676
- max_remote_readers
 - DDS_DataWriterResourceLimitsQosPolicy, 1679
- max_remote_virtual_writers
 - DDS_DataReaderResourceLimitsQosPolicy, 1630
- max_remote_virtual_writers_per_instance
 - DDS_DataReaderResourceLimitsQosPolicy, 1631
- max_remote_writers
 - DDS_DataReaderResourceLimitsQosPolicy, 1621
- max_remote_writers_per_instance
 - DDS_DataReaderResourceLimitsQosPolicy, 1622
- max_remote_writers_per_sample
 - DDS_DataReaderResourceLimitsQosPolicy, 1632
- max_samples
 - DDS_BatchQosPolicy, 1553
 - DDS_BuiltinTopicReaderResourceLimits_t, 1558
 - DDS_DurabilityServiceQosPolicy, 1766
 - DDS_ResourceLimitsQosPolicy, 1951
- max_samples_per_instance
 - DDS_DurabilityServiceQosPolicy, 1766
 - DDS_ResourceLimitsQosPolicy, 1952
- max_samples_per_read
 - DDS_BuiltinTopicReaderResourceLimits_t, 1560
 - DDS_DataReaderResourceLimitsQosPolicy, 1625
- max_samples_per_remote_writer

- DDS_DataReaderResourceLimitsQosPolicy, 1622
- max_send_window_size
 - DDS_RtpsReliableWriterProtocol_t, 1973
- max_size_serialized
 - DDS_DynamicDataTypeSerializationProperty_t, 1783
- max_skiplist_level
 - DDS_DatabaseQosPolicy, 1584
- max_tokens
 - DDS_FlowControllerTokenBucketProperty_t, 1801
- max_topic_queries
 - DDS_DataReaderResourceLimitsQosPolicy, 1634
- max_total_instances
 - DDS_DataReaderResourceLimitsQosPolicy, 1629
- max_virtual_writers
 - DDS_DataWriterResourceLimitsQosPolicy, 1679
- max_weak_references
 - DDS_DatabaseQosPolicy, 1584
- member_count
 - DDS_DynamicDataInfo, 1776
- member_exists
 - DDS_DynamicDataMemberInfo, 1778
- member_id
 - DDS_DynamicDataMemberInfo, 1777
- member_kind
 - DDS_DynamicDataMemberInfo, 1778
- member_name
 - DDS_DynamicDataMemberInfo, 1778
- message_auth
 - DDS_ParticipantTrustSignatureAlgorithmInfo, 1881
- message_id
 - NDDS_Config_LogMessage, 2119
- message_size_max
 - DDS_TransportInfo_t, 2053
 - NDDS_Transport_Property_t, 2124
- metatraffic_transport_priority
 - DDS_DiscoveryQosPolicy, 1717
- metrics
 - DDS_MonitoringTelemetryData, 1855
- Middleware Memory Management, 878
- middleware_forwarding_level
 - DDS_MonitoringLoggingForwardingSettings, 1847
- min_app_ack_response_keep_duration
 - DDS_RtpsReliableReaderProtocol_t, 1958
- min_heartbeat_response_delay
 - DDS_RtpsReliableReaderProtocol_t, 1955
- min_initial_participant_announcement_period
 - DDS_DiscoveryConfigQosPolicy, 1695
- min_nack_response_delay
 - DDS_RtpsReliableWriterProtocol_t, 1968
- min_send_window_size
 - DDS_RtpsReliableWriterProtocol_t, 1973
- min_size_serialized
 - DDS_DynamicDataTypeSerializationProperty_t, 1783
- minimum_separation
 - DDS_TimeBasedFilterQosPolicy, 2034
- minor
 - DDS_ProductVersion_t, 1897
 - DDS_ProtocolVersion_t, 1904
 - NDDS_Config_LibraryVersion_t, 2115
- MONITORING, 1299
 - DDS_MONITORING_QOS_POLICY_NAME, 1300
- monitoring
 - DDS_DomainParticipantFactoryQos, 1722
- Monitoring Library 2.0, 1434
 - RTI_MONITORING_EVENT_TOPIC_NAME, 1436
 - RTI_Monitoring_initialize, 1435
 - RTI_MONITORING_LOGGING_TOPIC_NAME, 1436
 - RTI_MONITORING_PERIODIC_TOPIC_NAME, 1436
- Multi-channel DataWriters, 719
- multi_channel
 - DDS_DataWriterQos, 1672
- multicast
 - DDS_DataReaderQos, 1616
- Multicast Mapping, 1351
- Multicast Settings, 1351
- multicast_enabled
 - NDDS_Transport_UDPv4_Property_t, 2137
 - NDDS_Transport_UDPv6_Property_t, 2157
- multicast_locators
 - DDS_SubscriptionBuiltinTopicData, 2018
- multicast_loopback_disabled
 - NDDS_Transport_UDPv4_Property_t, 2137
 - NDDS_Transport_UDPv6_Property_t, 2157
- multicast_mapping
 - DDS_DomainParticipantQos, 1730
- multicast_receive_addresses
 - DDS_DiscoveryQosPolicy, 1717
- multicast_resend_threshold
 - DDS_RtpsReliableWriterProtocol_t, 1977
- multicast_settings
 - DDS_ChannelSettings_t, 1564
- multicast_ttl
 - NDDS_Transport_UDPv4_Property_t, 2137
 - NDDS_Transport_UDPv6_Property_t, 2157
- MULTICHANNEL, 1300
 - DDS_MULTICHANNEL_QOS_POLICY_NAME, 1300
- Mutable QoS, 949
- nack_period
 - DDS_RtpsReliableReaderProtocol_t, 1956
- nack_suppression_duration
 - DDS_RtpsReliableWriterProtocol_t, 1969

- name
 - DDS_EntityNameQosPolicy, 1790
 - DDS_EnumMember, 1792
 - DDS_PartitionQosPolicy, 1883
 - DDS_Property_t, 1901
 - DDS_StructMember, 2007
 - DDS_Tag, 2026
 - DDS_TopicBuiltinTopicData, 2036
 - DDS_UnionMember, 2084
 - DDS_ValueMember, 2089
- nanosec
 - DDS_Duration_t, 1768
 - DDS_Time_t, 2031
- NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_DOMAIN_ID
 - Activity Context, 1462
- NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_ENTITY_KIND
 - Activity Context, 1462
- NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_ENTITY_NAME
 - Activity Context, 1462
- NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_GUID_PREFIX
 - Activity Context, 1461
- NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_ALL
 - Activity Context, 1460
- NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_DELETE
 - Activity Context, 1460
- NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_MASK_NONE
 - Activity Context, 1460
- NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_TOPIC
 - Activity Context, 1461
- NDDS_CONFIG_ACTIVITY_CONTEXT_ATTRIBUTE_TYPE
 - Activity Context, 1461
- NDDS_Config_ActivityContext_set_attribute_mask
 - Activity Context, 1462
- NDDS_Config_ActivityContextAttributeKind
 - Activity Context, 1460
- NDDS_Config_ActivityContextAttributeKindMask
 - Activity Context, 1460
- NDDS_Config_enable_automatic_tss_cleanup
 - Other Utilities, 1498
- NDDS_Config_get_rpc_compliance_mask
 - Compliance Configuration, 1475
- NDDS_Config_get_xtypes_compliance_mask
 - Compliance Configuration, 1474
- NDDS_Config_LibraryVersion_t, 2114
 - build, 2115
 - major, 2115
 - minor, 2115
 - release, 2115
- NDDS_Config_load_compliance_masks
 - Compliance Configuration, 1475
- NDDS_CONFIG_LOG_CATEGORY_ALL
 - Logging, 1446
- NDDS_CONFIG_LOG_CATEGORY_API
 - Logging, 1445
- NDDS_CONFIG_LOG_CATEGORY_COMMUNICATION
 - Logging, 1445
- NDDS_CONFIG_LOG_CATEGORY_DATABASE
 - Logging, 1445
- NDDS_CONFIG_LOG_CATEGORY_DISCOVERY
 - Logging, 1446
- NDDS_CONFIG_LOG_CATEGORY_ENTITIES
 - Logging, 1445
- NDDS_CONFIG_LOG_CATEGORY_PLATFORM
 - Logging, 1445
- NDDS_CONFIG_LOG_CATEGORY_SECURITY
 - Logging, 1446
- NDDS_CONFIG_LOG_CATEGORY_USER
 - Logging, 1446
- NDDS_CONFIG_LOG_FACILITY_MIDDLEWARE
 - Logging, 1447
- NDDS_CONFIG_LOG_FACILITY_SECURITY_EVENT
 - Logging, 1447
- NDDS_CONFIG_LOG_FACILITY_SERVICE
 - Logging, 1447
- NDDS_CONFIG_LOG_FACILITY_USER
 - Logging, 1447
- NDDS_CONFIG_LOG_LEVEL_DEBUG
 - Logging, 1444
- NDDS_CONFIG_LOG_LEVEL_ERROR
 - Logging, 1444
- NDDS_CONFIG_LOG_LEVEL_FATAL_ERROR
 - Logging, 1444
- NDDS_CONFIG_LOG_LEVEL_STATUS_LOCAL
 - Logging, 1444
- NDDS_CONFIG_LOG_LEVEL_STATUS_REMOTE
 - Logging, 1444
- NDDS_CONFIG_LOG_LEVEL_WARNING
 - Logging, 1444
- NDDS_CONFIG_LOG_PRINT_FORMAT_DEBUG
 - Logging, 1446
- NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT
 - Logging, 1446
- NDDS_CONFIG_LOG_PRINT_FORMAT_MAXIMAL
 - Logging, 1446
- NDDS_CONFIG_LOG_PRINT_FORMAT_MINIMAL
 - Logging, 1446
- NDDS_CONFIG_LOG_PRINT_FORMAT_TIMESTAMPED
 - Logging, 1446
- NDDS_CONFIG_LOG_PRINT_FORMAT_VERBOSE
 - Logging, 1446
- NDDS_CONFIG_LOG_PRINT_FORMAT_VERBOSE_TIMESTAMPED
 - Logging, 1446
- NDDS_CONFIG_LOG_VERBOSITY_ERROR
 - Logging, 1443
- NDDS_CONFIG_LOG_VERBOSITY_SILENT
 - Logging, 1443
- NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL
 - Logging, 1443

- NDDS_CONFIG_LOG_VERBOSITY_STATUS_LOCAL
 - Logging, 1443
- NDDS_CONFIG_LOG_VERBOSITY_STATUS_REMOTE
 - Logging, 1443
- NDDS_CONFIG_LOG_VERBOSITY_WARNING
 - Logging, 1443
- NDDS_Config_LogCategory
 - Logging, 1445
- NDDS_Config_LogFacility
 - Logging, 1446
- NDDS_Config_Logger, 2116
- NDDS_Config_Logger_alert
 - Logging, 1453
- NDDS_Config_Logger_critical
 - Logging, 1453
- NDDS_Config_Logger_debug
 - Logging, 1456
- NDDS_Config_Logger_emergency
 - Logging, 1452
- NDDS_Config_Logger_error
 - Logging, 1454
- NDDS_Config_Logger_get_instance
 - Logging, 1448
- NDDS_Config_Logger_get_instancel
 - Logging, 1449
- NDDS_Config_Logger_get_output_device
 - Logging, 1456
- NDDS_Config_Logger_get_output_file
 - Logging, 1450
- NDDS_Config_Logger_get_print_format
 - Logging, 1451
- NDDS_Config_Logger_get_print_format_by_log_level
 - Logging, 1452
- NDDS_Config_Logger_get_verbosity
 - Logging, 1449
- NDDS_Config_Logger_get_verbosity_by_category
 - Logging, 1449
- NDDS_Config_Logger_informational
 - Logging, 1455
- NDDS_Config_Logger_notice
 - Logging, 1455
- NDDS_Config_Logger_set_output_device
 - Logging, 1457
- NDDS_Config_Logger_set_output_file
 - Logging, 1450
- NDDS_Config_Logger_set_output_file_name
 - Logging, 1451
- NDDS_Config_Logger_set_output_file_set
 - Logging, 1451
- NDDS_Config_Logger_set_print_format
 - Logging, 1452
- NDDS_Config_Logger_set_print_format_by_log_level
 - Logging, 1452
- NDDS_Config_Logger_set_verbosity
 - Logging, 1450
- NDDS_Config_Logger_set_verbosity_by_category
 - Logging, 1450
- NDDS_Config_Logger_warning
 - Logging, 1454
- NDDS_Config_LoggerDevice, 2116
 - close, 2117
 - write, 2117
- NDDS_Config_LoggerDevice_INITIALIZER
 - Logging, 1442
- NDDS_Config_LoggerDeviceCloseFnc
 - Logging, 1443
- NDDS_Config_LoggerDeviceWriteFnc
 - Logging, 1442
- NDDS_Config_LogLevel
 - Logging, 1444
- NDDS_Config_LogMessage, 2117
 - facility, 2119
 - is_license_message, 2118
 - is_security_message, 2118
 - level, 2118
 - message_id, 2119
 - text, 2118
 - timestamp, 2119
- NDDS_Config_LogPrintFormat
 - Logging, 1446
- NDDS_Config_LogVerbosity
 - Logging, 1443
- NDDS_CONFIG_RPC_COMPLIANCE_MASK_DEFAULT
 - Compliance Configuration, 1465
- NDDS_CONFIG_RPC_COMPLIANCE_MASK_VENDOR
 - Compliance Configuration, 1465
- NDDS_CONFIG_RPC_USE_STANDARD_RELATED_SAMPLE_PID_BIT
 - Compliance Configuration, 1474
- NDDS_Config_RpcComplianceMask
 - Compliance Configuration, 1466
- NDDS_Config_RpcComplianceMaskBits
 - Compliance Configuration, 1473
- NDDS_Config_set_rpc_compliance_mask
 - Compliance Configuration, 1476
- NDDS_Config_set_xtypes_compliance_mask
 - Compliance Configuration, 1474
- NDDS_CONFIG_SYSLOG_LEVEL_ALERT
 - Logging, 1444
- NDDS_CONFIG_SYSLOG_LEVEL_CRITICAL
 - Logging, 1445
- NDDS_CONFIG_SYSLOG_LEVEL_DEBUG
 - Logging, 1445
- NDDS_CONFIG_SYSLOG_LEVEL_EMERGENCY
 - Logging, 1444
- NDDS_CONFIG_SYSLOG_LEVEL_ERROR
 - Logging, 1445
- NDDS_CONFIG_SYSLOG_LEVEL_INFORMATIONAL
 - Logging, 1445

- NDDS_CONFIG_SYSLOG_LEVEL_NOTICE
 - Logging, 1445
- NDDS_CONFIG_SYSLOG_LEVEL_WARNING
 - Logging, 1445
- NDDS_CONFIG_SYSLOG_VERBOSITY_ALERT
 - Logging, 1448
- NDDS_CONFIG_SYSLOG_VERBOSITY_CRITICAL
 - Logging, 1448
- NDDS_CONFIG_SYSLOG_VERBOSITY_DEBUG
 - Logging, 1448
- NDDS_CONFIG_SYSLOG_VERBOSITY_EMERGENCY
 - Logging, 1447
- NDDS_CONFIG_SYSLOG_VERBOSITY_ERROR
 - Logging, 1448
- NDDS_CONFIG_SYSLOG_VERBOSITY_INFORMATIONal
 - Logging, 1448
- NDDS_CONFIG_SYSLOG_VERBOSITY_NOTICE
 - Logging, 1448
- NDDS_CONFIG_SYSLOG_VERBOSITY_SILENT
 - Logging, 1447
- NDDS_CONFIG_SYSLOG_VERBOSITY_WARNING
 - Logging, 1448
- NDDS_Config_SyslogLevel
 - Logging, 1444
- NDDS_Config_SyslogVerbosity
 - Logging, 1447
- NDDS_Config_Version_get_api_version
 - Version, 1437
- NDDS_Config_Version_get_core_version
 - Version, 1438
- NDDS_Config_Version_get_product_version
 - Version, 1437
- NDDS_Config_Version_t, 2119
- NDDS_Config_Version_to_string
 - Version, 1438
- NDDS_CONFIG_XTYPES_ACCEPT_UNKNOWN_DISCRIMINATOR_BIT
 - Compliance Configuration, 1470
- NDDS_CONFIG_XTYPES_ACCEPT_UNKNOWN_ENUM_VALUES_BIT
 - Compliance Configuration, 1469
- NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_DEFAULT
 - Compliance Configuration, 1464
- NDDS_CONFIG_XTYPES_COMPLIANCE_MASK_VENDOR
 - Compliance Configuration, 1464
- NDDS_CONFIG_XTYPES_DHEADER_IN_NON_PRIMITIVE_COLLECTIONS_BIT
 - Compliance Configuration, 1468
- NDDS_CONFIG_XTYPES_ENCAPSULATION_OPTIONS_WITH_TRANSPORT_BIT
 - Compliance Configuration, 1469
- NDDS_CONFIG_XTYPES_ENUM_AS_PRIMITIVE_IN_COLLECTIONS_BIT
 - Compliance Configuration, 1468
- NDDS_CONFIG_XTYPES_INITIALIZE_DISCRIMINATOR_TO_DEFAULT_BIT
 - Compliance Configuration, 1472
- NDDS_CONFIG_XTYPES_JAVA_KEYHASH_750_BIT
 - Compliance Configuration, 1473
- NDDS_CONFIG_XTYPES_PARAMETER_LENGTH_WITH_PADDING_BIT
 - Compliance Configuration, 1468
- NDDS_CONFIG_XTYPES_PYTHON_INT8_730_BIT
 - Compliance Configuration, 1473
- NDDS_CONFIG_XTYPES_SELECT_DEFAULT_DISCRIMINATOR_BIT
 - Compliance Configuration, 1470
- NDDS_CONFIG_XTYPES_SENTINEL_IN_EMPTY_UNION_BIT
 - Compliance Configuration, 1471
- NDDS_Config_XTypesComplianceMask
 - Compliance Configuration, 1465
- NDDS_Config_XTypesComplianceMaskBits
 - Compliance Configuration, 1467
- NDDS_DISCOVERY_PEERS, 1352
- NDDS_LM_unload_activation_library
 - Other Utilities, 1498
- NDDS_Transport_Address_from_string
 - Transport Address, 1030
- NDDS_TRANSPORT_ADDRESS_INVALID
 - Transport Address, 1032
- NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER
 - Transport Address, 1028
- NDDS_Transport_Address_is_ipv4
 - Transport Address, 1031
- NDDS_Transport_Address_is_multicast
 - Transport Address, 1032
- NDDS_Transport_Address_print
 - Transport Address, 1031
- NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE
 - Transport Address, 1029
- NDDS_Transport_Address_t, 2120
 - network_ordered_value, 2120
- NDDS_Transport_Address_to_string
 - Transport Address, 1029
- NDDS_Transport_Address_to_string_with_protocol_family_format
 - Transport Address, 1030
- NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT
 - Transport Plugins Configuration, 1023
- NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL_COUNT
 - Transport Plugins Configuration, 1023
- NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_COUNT_UNLIMITED
 - Transport Plugins Configuration, 1022
- NDDS_TRANSPORT_CLASSID_INVALID
 - Transport Plugins Configuration, 1023
- NDDS_TRANSPORT_CLASSID_RESERVED_RANGE
 - Transport Plugins Configuration, 1025
- NDDS_TRANSPORT_CLASSID_SHMEM
 - Transport Plugins Configuration, 1024
- NDDS_TRANSPORT_CLASSID_SHMEM_510
 - Transport Plugins Configuration, 1024
- NDDS_Transport_ClassId_t
 - Transport Plugins Configuration, 1026
- NDDS_TRANSPORT_CLASSID_TCPV4_LAN
 - Transport Plugins Configuration, 1024
- NDDS_TRANSPORT_CLASSID_TCPV4_WAN
 - Transport Plugins Configuration, 1024

- NDDS_TRANSPORT_CLASSID_TLSV4_LAN
 - Transport Plugins Configuration, 1025
- NDDS_TRANSPORT_CLASSID_TLSV4_WAN
 - Transport Plugins Configuration, 1025
- NDDS_TRANSPORT_CLASSID_UDPv4
 - Transport Plugins Configuration, 1023
- NDDS_TRANSPORT_CLASSID_UDPv4_WAN
 - Transport Plugins Configuration, 1025
- NDDS_TRANSPORT_CLASSID_UDPv6
 - Transport Plugins Configuration, 1024
- NDDS_TRANSPORT_CLASSID_UDPv6_510
 - Transport Plugins Configuration, 1024
- NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN
 - Transport Plugins Configuration, 1025
- NDDS_Transport_create_plugin
 - Installing Transport Plugins, 729
- NDDS_Transport_Handle_is_nil
 - Installing Transport Plugins, 730
- NDDS_TRANSPORT_HANDLE_NIL
 - Installing Transport Plugins, 736
- NDDS_Transport_Handle_t
 - Installing Transport Plugins, 729
- NDDS_TRANSPORT_INTERFACE_OFF
 - Interface, 1020
- NDDS_TRANSPORT_INTERFACE_ON
 - Interface, 1020
- NDDS_TRANSPORT_INTERFACE_RANK_UNKNOWN
 - Transport Plugins Configuration, 1022
- NDDS_Transport_Interface_Status_t
 - Interface, 1019
- NDDS_Transport_Interface_t, 2120
 - address, 2121
 - rank, 2121
 - status, 2121
 - transport_classid, 2121
- NDDS_TRANSPORT_LENGTH_UNLIMITED
 - Transport Plugins Configuration, 1022
- NDDS_TRANSPORT_PORT_INVALID
 - Transport Plugins Configuration, 1022
- NDDS_Transport_Port_t
 - Transport Plugins Configuration, 1026
- NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOAD
 - Transport Plugins Configuration, 1025
- NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_COUNT_MAX
 - Transport Plugins Configuration, 1026
- NDDS_Transport_Property_t, 2122
 - address_bit_count, 2123
 - allow_interfaces_list, 2125
 - allow_interfaces_list_length, 2125
 - allow_multicast_interfaces_list, 2126
 - allow_multicast_interfaces_list_length, 2127
 - classid, 2123
 - deny_interfaces_list, 2126
 - deny_interfaces_list_length, 2126
 - deny_multicast_interfaces_list, 2127
 - deny_multicast_interfaces_list_length, 2127
 - gather_send_buffer_count_max, 2124
 - max_interface_count, 2128
 - message_size_max, 2124
 - properties_bitmap, 2124
 - thread_name_prefix, 2128
 - transport_uuid, 2128
- NDDS_TRANSPORT_SHMEM_ADDRESS_BIT_COUNT
 - Shared Memory Transport, 1039
- NDDS_Transport_Shmem_create
 - Shared Memory Transport, 1041
- NDDS_TRANSPORT_SHMEM_GATHER_SEND_BUFFER_COUNT_MAX
 - Shared Memory Transport, 1039
- NDDS_TRANSPORT_SHMEM_MAJOR_AFTER_BUG_14240_FIX
 - Shared Memory Transport, 1040
- NDDS_TRANSPORT_SHMEM_MESSAGE_SIZE_MAX_DEFAULT
 - Shared Memory Transport, 1039
- NDDS_Transport_Shmem_new
 - Shared Memory Transport, 1040
- NDDS_TRANSPORT_SHMEM_PROPERTIES_BITMAP_DEFAULT
 - Shared Memory Transport, 1039
- NDDS_TRANSPORT_SHMEM_PROPERTY_DEFAULT
 - Shared Memory Transport, 1040
- NDDS_Transport_Shmem_Property_t, 2129
 - enable_udp_debugging, 2131
 - parent, 2130
 - receive_buffer_size, 2130
 - received_message_count_max, 2130
 - udp_debugging_address, 2131
 - udp_debugging_port, 2132
- NDDS_TRANSPORT_SHMEM_RECEIVE_BUFFER_SIZE_DEFAULT
 - Shared Memory Transport, 1040
- NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX_DEFAULT
 - Shared Memory Transport, 1040
- NDDS_Transport_Support, 2132
 - NDDS_Transport_Support_add_receive_route
 - Installing Transport Plugins, 733
 - NDDS_Transport_Support_add_send_route
 - Installing Transport Plugins, 732
 - NDDS_Transport_Support_get_builtin_transport_property
 - Installing Transport Plugins, 734
 - NDDS_Transport_Support_get_transport_plugin
 - Installing Transport Plugins, 735
 - NDDS_Transport_Support_lookup_transport
 - Installing Transport Plugins, 732
 - NDDS_Transport_Support_register_transport
 - Installing Transport Plugins, 730
 - NDDS_Transport_Support_set_builtin_transport_property
 - Installing Transport Plugins, 735
- NDDS_TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_MAX_DEFAULT
 - UDP Transport Plugin definitions, 1034
- NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT
 - UDP Transport Plugin definitions, 1035

- NDDS_Transport_UDP_Port
 - UDP Transport Plugin definitions, 1035
- NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT
 - UDP Transport Plugin definitions, 1033
- NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT
 - UDP Transport Plugin definitions, 1035
- NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT
 - UDP Transport Plugin definitions, 1034
- NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT
 - UDP Transport Plugin definitions, 1034
- NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_MAX
 - UDP Transport Plugin definitions, 1034
- NDDS_Transport_UDP_WAN_CommPortsMappingInfo, 2132
 - host_port, 2133
 - public_port, 2133
 - rtps_port, 2133
- NDDS_TRANSPORT_UDPv4_ADDRESS_BIT_COUNT
 - UDPv4 Transport, 1046
- NDDS_TRANSPORT_UDPv4_BLOCKING_ALWAYS
 - UDPv4 Transport, 1048
- NDDS_TRANSPORT_UDPv4_BLOCKING_DEFAULT
 - UDPv4 Transport, 1048
- NDDS_TRANSPORT_UDPv4_BLOCKING_NEVER
 - UDPv4 Transport, 1048
- NDDS_Transport_UDPv4_create
 - UDPv4 Transport, 1050
- NDDS_Transport_UDPv4_create_from_properties_with_prefix
 - UDPv4 Transport, 1051
- NDDS_TRANSPORT_UDPv4_GATHER_SEND_BUFFER_SIZE_MAX_DEFAULT
 - UDPv4 Transport, 1046
- NDDS_TRANSPORT_UDPv4_MESSAGE_SIZE_MAX_DEFAULT
 - UDPv4 Transport, 1047
- NDDS_TRANSPORT_UDPv4_MULTICAST_TTL_DEFAULT
 - UDPv4 Transport, 1048
- NDDS_Transport_UDPv4_new
 - UDPv4 Transport, 1049
- NDDS_TRANSPORT_UDPv4_PAYLOAD_SIZE_MAX
 - UDPv4 Transport, 1047
- NDDS_TRANSPORT_UDPv4_PROPERTIES_BITMAP_DEFAULT
 - UDPv4 Transport, 1046
- NDDS_TRANSPORT_UDPv4_PROPERTY_DEFAULT
 - UDPv4 Transport, 1048
- NDDS_Transport_UDPv4_Property_t, 2133
 - disable_interface_tracking, 2142
 - force_interface_poll_detection, 2141
 - ignore_loopback_interface, 2137
 - ignore_nonrunning_interfaces, 2138
 - ignore_nonup_interfaces, 2138
 - interface_poll_period, 2141
 - join_multicast_group_timeout, 2142
 - multicast_enabled, 2137
 - multicast_loopback_disabled, 2137
 - multicast_ttl, 2137
 - no_zero_copy, 2139
 - parent, 2135
 - port_offset, 2143
 - protocol_overhead_max, 2142
 - public_address, 2143
 - recv_socket_buffer_size, 2136
 - enable_multicast_receive_resource, 2141
 - send_blocking, 2139
 - allow_ping, 2141
 - send_socket_buffer_size, 2135
 - transport_priority_mapping_high, 2140
 - transport_priority_mapping_low, 2140
 - transport_priority_mask, 2140
 - unicast_enabled, 2136
 - use_checksum, 2139
- NDDS_TRANSPORT_UDPv4_RECV_SOCKET_BUFFER_SIZE_DEFAULT
 - UDPv4 Transport, 1047
- NDDS_TRANSPORT_UDPv4_SEND_SOCKET_BUFFER_SIZE_DEFAULT
 - UDPv4 Transport, 1047
- NDDS_TRANSPORT_UDPv4_SOCKET_BUFFER_SIZE_OS_DEFAULT
 - UDPv4 Transport, 1047
- NDDS_TRANSPORT_UDPv4_SOCKET_BUFFER_SIZE_OS_MAX
 - UDPv4 Transport, 1046
- NDDS_Transport_UDPv4_string_to_address_cEA
 - UDPv4 Transport, 1048
- NDDS_TRANSPORT_UDPv4_WAN_ADDRESS_BIT_COUNT
 - UDPv4 Transport, 1046
- NDDS_Transport_UDPv4_WAN_create
 - Real-Time WAN Transport, 1056
- NDDS_Transport_UDPv4_WAN_create_from_properties_with_prefix
 - Real-Time WAN Transport, 1057
- NDDS_TRANSPORT_UDPv4_WAN_MESSAGE_SIZE_MAX_DEFAULT
 - Real-Time WAN Transport, 1055
- NDDS_Transport_UDPv4_WAN_new
 - Real-Time WAN Transport, 1056
- NDDS_TRANSPORT_UDPv4_WAN_PROPERTY_DEFAULT
 - Real-Time WAN Transport, 1055
- NDDS_Transport_UDPv4_WAN_Property_t, 2144
 - binding_ping_period, 2153
 - comm_ports_list, 2151
 - comm_ports_list_length, 2152
 - disable_interface_tracking, 2150
 - force_interface_poll_detection, 2150
 - ignore_loopback_interface, 2146
 - ignore_nonrunning_interfaces, 2147
 - ignore_nonup_interfaces, 2146
 - interface_poll_period, 2150
 - no_zero_copy, 2147
 - parent, 2145
 - port_offset, 2153
 - protocol_overhead_max, 2150
 - public_address, 2151
 - recv_socket_buffer_size, 2146
 - send_blocking, 2148

- send_ping, 2149
- send_socket_buffer_size, 2145
- transport_priority_mapping_high, 2149
- transport_priority_mapping_low, 2149
- transport_priority_mask, 2148
- use_checksum, 2148
- NDDS_TRANSPORT_UDPV6_ADDRESS_BIT_COUNT
 - UDpv6 Transport, 1062
- NDDS_TRANSPORT_UDPV6_BLOCKING_ALWAYS
 - UDpv6 Transport, 1064
- NDDS_TRANSPORT_UDPV6_BLOCKING_NEVER
 - UDpv6 Transport, 1064
- NDDS_Transport_UDPv6_create
 - UDpv6 Transport, 1066
- NDDS_Transport_UDPv6_create_from_properties_with_prefix
 - UDpv6 Transport, 1067
- NDDS_TRANSPORT_UDPV6_GATHER_SEND_BUFFER_SIZE_MAX_DEFAULT
 - UDpv6 Transport, 1062
- NDDS_TRANSPORT_UDPV6_MESSAGE_SIZE_MAX_DEFAULT
 - UDpv6 Transport, 1063
- NDDS_TRANSPORT_UDPV6_MULTICAST_TTL_DEFAULT
 - UDpv6 Transport, 1064
- NDDS_Transport_UDPv6_new
 - UDpv6 Transport, 1065
- NDDS_TRANSPORT_UDPV6_PAYLOAD_SIZE_MAX
 - UDpv6 Transport, 1063
- NDDS_TRANSPORT_UDPV6_PROPERTIES_BITMAP_DEFAULT
 - UDpv6 Transport, 1062
- NDDS_Transport_UDPv6_Property_t, 2154
 - disable_interface_tracking, 2162
 - enable_v4mapped, 2159
 - force_interface_poll_detection, 2161
 - ignore_loopback_interface, 2157
 - ignore_nonrunning_interfaces, 2158
 - interface_poll_period, 2161
 - join_multicast_group_timeout, 2162
 - multicast_enabled, 2157
 - multicast_loopback_disabled, 2157
 - multicast_ttl, 2157
 - no_zero_copy, 2158
 - parent, 2155
 - port_offset, 2163
 - protocol_overhead_max, 2161
 - public_address, 2162
 - recv_socket_buffer_size, 2156
 - reuse_multicast_receive_resource, 2161
 - send_blocking, 2159
 - send_ping, 2160
 - send_socket_buffer_size, 2155
 - transport_priority_mapping_high, 2160
 - transport_priority_mapping_low, 2160
 - transport_priority_mask, 2159
 - unicast_enabled, 2156
- NDDS_TRANSPORT_UDPV6_RECV_SOCKET_BUFFER_SIZE_DEFAULT
 - UDpv6 Transport, 1063
- NDDS_TRANSPORT_UDPV6_SEND_SOCKET_BUFFER_SIZE_DEFAULT
 - UDpv6 Transport, 1063
- NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT
 - UDpv6 Transport, 1063
- NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX
 - UDpv6 Transport, 1062
- NDDS_Transport_UDPv6_string_to_address_cEA
 - UDpv6 Transport, 1064
- NDDS_Transport_UUID, 2164
- NDDS_TRANSPORT_UUID_SIZE
 - Transport Plugins Configuration, 1022
- NDDS_TRANSPORT_UUID_UNKNOWN
 - Transport Plugins Configuration, 1022
- NDDS_Utility_disable_heap_monitoring
 - Heap Monitoring, 1480
- NDDS_Utility_disable_network_capture
 - Network Capture, 1490
- NDDS_Utility_enable_heap_monitoring
 - Heap Monitoring, 1479
- NDDS_Utility_enable_heap_monitoring_w_params
 - Heap Monitoring, 1480
- NDDS_Utility_enable_network_capture
 - Network Capture, 1489
- NDDS_Utility_get_spin_per_microsecond
 - Other Utilities, 1500
- NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT_ACTUAL
 - Heap Monitoring, 1479
- NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT_FUNCTIONS
 - Heap Monitoring, 1479
- NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_BIT_TOP
 - Heap Monitoring, 1479
- NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_DEFAULT
 - Heap Monitoring, 1479
- NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_CONTENT_MINIMAL
 - Heap Monitoring, 1479
- NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_OUTPUT_FORMAT_C
 - Heap Monitoring, 1479
- NDDS_UTILITY_HEAP_MONITORING_SNAPSHOT_OUTPUT_FORMAT_S
 - Heap Monitoring, 1479
- NDDS_Utility_HeapMonitoringParams_INITIALIZER
 - Heap Monitoring, 1478
- NDDS_Utility_HeapMonitoringParams_t, 2164
 - Heap Monitoring, 1478
 - snapshot_content_format, 2164
 - snapshot_output_format, 2164
- NDDS_Utility_HeapMonitoringSnapshotContentFormat
 - Heap Monitoring, 1479
- NDDS_Utility_HeapMonitoringSnapshotOutputFormat
 - Heap Monitoring, 1478
- NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_ENCRYPTED_DATA
 - Network Capture, 1489
- NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_ALL
 - Network Capture, 1487

- NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_NONE, 1486
- Network Capture, 1486
- NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_NONE, 1486
- Network Capture, 1486
- NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_USER_SERIALIZE_DATA, 1489
- Network Capture, 1489
- NDDS_UTILITY_NETWORK_CAPTURE_PARAMETERS_DEFAULT, 1497
- Network Capture, 1497
- NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_IN, 1489
- Network Capture, 1489
- NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_ALL, 1487
- Network Capture, 1487
- NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_DEFAULT, 1487
- Network Capture, 1487
- NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_NONE, 1487
- Network Capture, 1487
- NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_OUT, 1489
- Network Capture, 1489
- NDDS_Utility_NetworkCaptureContentKind, 1488
- Network Capture, 1488
- NDDS_Utility_NetworkCaptureContentKindMask, 1488
- Network Capture, 1488
- NDDS_Utility_NetworkCaptureParams_t, 2165
 - checkpoint_thread_settings, 2166
 - dropped_content, 2166
 - frame_queue_size, 2167
 - Network Capture, 1488
 - parse_encrypted_content, 2166
 - traffic, 2166
 - transports, 2165
- NDDS_Utility_NetworkCaptureTrafficKind, 1489
- Network Capture, 1489
- NDDS_Utility_NetworkCaptureTrafficKindMask, 1488
- Network Capture, 1488
- NDDS_Utility_pause_heap_monitoring, 1480
- Heap Monitoring, 1480
- NDDS_Utility_pause_network_capture, 1495
- Network Capture, 1495
- NDDS_Utility_pause_network_capture_for_participant, 1495
- Network Capture, 1495
- NDDS_Utility_resume_heap_monitoring, 1481
- Heap Monitoring, 1481
- NDDS_Utility_resume_network_capture, 1496
- Network Capture, 1496
- NDDS_Utility_resume_network_capture_for_participant, 1496
- Network Capture, 1496
- NDDS_Utility_set_default_network_capture_params, 1490
- Network Capture, 1490
- NDDS_Utility_sleep, 1499
- Other Utilities, 1499
- NDDS_Utility_spin, 1499
- Other Utilities, 1499
- NDDS_Utility_start_network_capture, 1491
- Network Capture, 1491
- NDDS_Utility_start_network_capture_for_participant, 1492
- Network Capture, 1492
- NDDS_Utility_start_network_capture_w_params, 1492
- Network Capture, 1492
- NDDS_Utility_start_network_capture_w_params_for_participant, 1493
- Network Capture, 1493
- NDDS_Utility_stop_network_capture, 1494
- Network Capture, 1494
- NDDS_Utility_stop_network_capture_for_participant, 1494
- Network Capture, 1494
- NDDS_Utility_take_heap_snapshot, 1481
- Heap Monitoring, 1481
- NDDS_Utility_disable_network_capture, 1490
- Network Capture, 1483
- NDDS_Utility_enable_network_capture, 1489
- Network Capture, 1483
- NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_ENCRYPTED_DATA, 1489
- NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_ALL, 1487
- NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_DEFAULT, 1486
- NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_MASK_NONE, 1486
- NDDS_UTILITY_NETWORK_CAPTURE_CONTENT_USER_SERIALIZE_DATA, 1489
- NDDS_UTILITY_NETWORK_CAPTURE_PARAMETERS_DEFAULT, 1497
- NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_IN, 1489
- NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_ALL, 1487
- NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_DEFAULT, 1487
- NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_MASK_NONE, 1487
- NDDS_UTILITY_NETWORK_CAPTURE_TRAFFIC_OUT, 1489
- NDDS_Utility_NetworkCaptureContentKind, 1488
- NDDS_Utility_NetworkCaptureContentKindMask, 1488
- NDDS_Utility_NetworkCaptureParams_t, 1488
- NDDS_Utility_NetworkCaptureTrafficKind, 1489
- NDDS_Utility_NetworkCaptureTrafficKindMask, 1488
- NDDS_Utility_pause_network_capture, 1495
- NDDS_Utility_pause_network_capture_for_participant, 1495
- NDDS_Utility_resume_network_capture, 1496
- NDDS_Utility_resume_network_capture_for_participant, 1496
- NDDS_Utility_set_default_network_capture_params, 1490
- NDDS_Utility_sleep, 1499
- NDDS_Utility_spin, 1499
- NDDS_Utility_start_network_capture, 1491
- NDDS_Utility_start_network_capture_for_participant, 1492

- NDDS_Utility_start_network_capture_w_params, 1492
 - NDDS_Utility_start_network_capture_w_params_for_participant, 1493
 - NDDS_Utility_stop_network_capture, 1494
 - NDDS_Utility_stop_network_capture_for_participant, 1494
- network_ordered_value
 - NDDS_Transport_Address_t, 2120
- new_remote_participant_announcements
 - DDS_DiscoveryConfigQosPolicy, 1695
- no_writers_generation_count
 - DDS_SampleInfo, 1990
- no_writers_instance_count
 - DDS_DataReaderCacheStatus, 1591
- no_writers_instance_count_peak
 - DDS_DataReaderCacheStatus, 1591
- no_writers_instance_removal
 - DDS_DataReaderResourceLimitsInstanceReplacementSettings, 1619
- no_zero_copy
 - NDDS_Transport_UDPv4_Property_t, 2139
 - NDDS_Transport_UDPv4_WAN_Property_t, 2147
 - NDDS_Transport_UDPv6_Property_t, 2158
- not_alive_count
 - DDS_LivelinessChangedStatus, 1821
- not_alive_count_change
 - DDS_LivelinessChangedStatus, 1821
- Observability Framework, 745
- Octet Buffer Support, 1500
 - DDS_OctetBuffer_alloc, 1502
 - DDS_OctetBuffer_dup, 1502
 - DDS_OctetBuffer_free, 1503
- Octets Built-in Type, 1151
 - DDS_Octets, 1154
 - DDS_Octets_delete, 1156
 - DDS_Octets_new, 1155
 - DDS_Octets_new_w_size, 1155
 - DDS_OctetsDataReader, 1154
 - DDS_OctetsDataReader_as_datareader, 1166
 - DDS_OctetsDataReader_narrow, 1166
 - DDS_OctetsDataReader_read, 1166
 - DDS_OctetsDataReader_read_next_sample, 1167
 - DDS_OctetsDataReader_read_w_condition, 1167
 - DDS_OctetsDataReader_return_loan, 1168
 - DDS_OctetsDataReader_take, 1166
 - DDS_OctetsDataReader_take_next_sample, 1168
 - DDS_OctetsDataReader_take_w_condition, 1167
 - DDS_OctetsDataWriter, 1154
 - DDS_OctetsDataWriter_as_datawriter, 1160
 - DDS_OctetsDataWriter_create_data, 1161
 - DDS_OctetsDataWriter_delete_data, 1161
 - DDS_OctetsDataWriter_narrow, 1160
 - DDS_OctetsDataWriter_write, 1161
 - DDS_OctetsDataWriter_write_octets, 1162
 - DDS_OctetsDataWriter_write_octets_seq, 1162
 - DDS_OctetsDataWriter_write_octets_seq_w_params, 1165
 - DDS_OctetsDataWriter_write_octets_seq_w_timestamp, 1164
 - DDS_OctetsDataWriter_write_octets_w_params, 1164
 - DDS_OctetsDataWriter_write_octets_w_timestamp, 1163
 - DDS_OctetsDataWriter_write_w_params, 1164
 - DDS_OctetsDataWriter_write_w_timestamp, 1163
 - DDS_OctetsTypeSupport_data_to_string, 1160
 - DDS_OctetsTypeSupport_deserialize_data_from_cdr_buffer, 1159
 - DDS_OctetsTypeSupport_get_type_name, 1158
 - DDS_OctetsTypeSupport_get_typecode, 1158
 - DDS_OctetsTypeSupport_print_data, 1158
 - DDS_OctetsTypeSupport_register_type, 1156
 - DDS_OctetsTypeSupport_serialize_data_to_cdr_buffer, 1159
 - DDS_OctetsTypeSupport_serialize_data_to_cdr_buffer_ex, 1159
 - DDS_OctetsTypeSupport_unregister_type, 1157
 - length, 1168
 - value, 1169
- old_source_timestamp_dropped_sample_count
 - DDS_DataReaderCacheStatus, 1587
- on_application_acknowledgment
 - DDS_DataWriterListener, 1648
- on_condition_triggered
 - DDS_ConditionHandler, 1571
- on_data_available
 - DDS_DataReaderListener, 1595
- on_data_on_readers
 - DDS_SubscriberListener, 2010
- on_empty_virtual_writer
 - DDS_DataWriterListener, 1648
- on_inconsistent_topic
 - DDS_TopicListener, 2042
- on_instance_replaced
 - DDS_DataWriterListener, 1647
- on_invalid_local_status_advance_notice
 - DDS_DomainParticipantListener, 1725
- on_liveliness_changed
 - DDS_DataReaderListener, 1594
- on_liveliness_lost
 - DDS_DataWriterListener, 1646
- on_new_sample
 - DDS_SampleHandler, 1984
- on_offered_deadline_missed
 - DDS_DataWriterListener, 1645
- on_offered_incompatible_qos

- DDS_DataWriterListener, 1645
- on_publication_matched
 - DDS_DataWriterListener, 1646
- on_reliable_reader_activity_changed
 - DDS_DataWriterListener, 1646
- on_reliable_writer_cache_changed
 - DDS_DataWriterListener, 1646
- on_request_available
 - Replier, 774, 775
- on_requested_deadline_missed
 - DDS_DataReaderListener, 1594
- on_requested_incompatible_qos
 - DDS_DataReaderListener, 1594
- on_sample_lost
 - DDS_DataReaderListener, 1595
- on_sample_rejected
 - DDS_DataReaderListener, 1594
- on_sample_removed
 - DDS_DataWriterListener, 1647
- on_service_request_accepted
 - DDS_DataWriterListener, 1648
- on_subscription_matched
 - DDS_DataReaderListener, 1595
- on_thread_deleted
 - DDS_AsyncWaitSetListener, 1544
- on_thread_spawned
 - DDS_AsyncWaitSetListener, 1544
- on_wait_timeout
 - DDS_AsyncWaitSetListener, 1544
- ordered_access
 - DDS_PresentationQosPolicy, 1893
- ordinal
 - DDS_EnumMember, 1792
- original_publication_virtual_guid
 - DDS_SampleInfo, 1993
- original_publication_virtual_sequence_number
 - DDS_SampleInfo, 1993
- original_related_reader_guid
 - DDS_TopicQueryData, 2047
- Other QoS Attributes, 956
- Other Utilities, 1498
 - NDDS_Config_enable_automatic_tss_cleanup, 1498
 - NDDS_LM_unload_activation_library, 1498
 - NDDS_Utility_get_spin_per_microsecond, 1500
 - NDDS_Utility_sleep, 1499
 - NDDS_Utility_spin, 1499
- out_of_range_rejected_sample_count
 - DDS_DataReaderProtocolStatus, 1609
- output_file
 - DDS_LoggingQosPolicy, 1832
- output_file_suffix
 - DDS_LoggingQosPolicy, 1833
- outstanding_asynchronous_sample_allocation
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1740
- OWNERSHIP, 1301
 - DDS_EXCLUSIVE_OWNERSHIP_QOS, 1302
 - DDS_OWNERSHIP_QOS_POLICY_NAME, 1302
 - DDS_OwnershipQosPolicyKind, 1301
 - DDS_SHARED_OWNERSHIP_QOS, 1302
- ownership
 - DDS_DataReaderQos, 1614
 - DDS_DataWriterQos, 1670
 - DDS_PublicationBuiltinTopicData, 1909
 - DDS_SubscriptionBuiltinTopicData, 2016
 - DDS_TopicBuiltinTopicData, 2038
 - DDS_TopicQos, 2045
- ownership_dropped_sample_count
 - DDS_DataReaderCacheStatus, 1588
- OWNERSHIP_STRENGTH, 1302
 - DDS_OWNERSHIPSTRENGTH_QOS_POLICY_NAME, 1303
- ownership_strength
 - DDS_DataWriterQos, 1670
 - DDS_PublicationBuiltinTopicData, 1909
- parent
 - NDDS_Transport_Shmem_Property_t, 2130
 - NDDS_Transport_UDPv4_Property_t, 2135
 - NDDS_Transport_UDPv4_WAN_Property_t, 2145
 - NDDS_Transport_UDPv6_Property_t, 2155
- parse_encrypted_content
 - NDDS_Utility_NetworkCaptureParams_t, 2166
- partial_configuration
 - DDS_ParticipantBuiltinTopicData, 1875
- participant
 - RTI_Connext_ReplierParams, 2170
 - RTI_Connext_RequesterParams, 2174
 - RTI_Connext_SimpleReplierParams, 2178
- Participant Built-in Topics, 1091
 - DDS_PARTICIPANT_TOPIC_NAME, 1092
 - DDS_ParticipantBuiltinTopicData, 1091
 - DDS_ParticipantBuiltinTopicDataDataReader, 1092
- Participant Use Cases, 975
- participant_announcement_period
 - DDS_DiscoveryConfigQosPolicy, 1693
- participant_configuration_reader
 - DDS_DiscoveryConfigQosPolicy, 1714
- participant_configuration_reader_resource_limits
 - DDS_DiscoveryConfigQosPolicy, 1714
- participant_configuration_writer
 - DDS_DiscoveryConfigQosPolicy, 1712
- participant_configuration_writer_data_lifecycle
 - DDS_DiscoveryConfigQosPolicy, 1713
- participant_configuration_writer_publish_mode
 - DDS_DiscoveryConfigQosPolicy, 1712
- participant_id

- DDS_WireProtocolQosPolicy, 2097
- participant_id_gain
 - DDS_RtpsWellKnownPorts_t, 1982
- participant_key
 - DDS_PublicationBuiltinTopicData, 1907
 - DDS_SubscriptionBuiltinTopicData, 2014
- participant_liveliness_assert_period
 - DDS_DiscoveryConfigQosPolicy, 1692
- participant_liveliness_lease_duration
 - DDS_DiscoveryConfigQosPolicy, 1692
- participant_message_reader
 - DDS_DiscoveryConfigQosPolicy, 1702
- participant_message_reader_reliability_kind
 - DDS_DiscoveryConfigQosPolicy, 1701
- participant_message_writer
 - DDS_DiscoveryConfigQosPolicy, 1702
- participant_name
 - DDS_DomainParticipantConfigParams_t, 1720
 - DDS_DomainParticipantQos, 1730
 - DDS_ParticipantBuiltinTopicData, 1873
- participant_property_list_max_length
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1752
- participant_property_string_max_length
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1753
- participant_qos_library_name
 - DDS_DomainParticipantConfigParams_t, 1720
- participant_qos_profile_name
 - DDS_DomainParticipantConfigParams_t, 1720
 - DDS_MonitoringDedicatedParticipantSettings, 1837
- participant_reader_resource_limits
 - DDS_DiscoveryConfigQosPolicy, 1696
- participant_user_data_max_length
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1746
- PARTITION, 1303
 - DDS_PARTITION_QOS_POLICY_NAME, 1303
- partition
 - DDS_DomainParticipantQos, 1731
 - DDS_ParticipantBuiltinTopicData, 1874
 - DDS_PublicationBuiltinTopicData, 1910
 - DDS_PublisherQos, 1919
 - DDS_SubscriberQos, 2011
 - DDS_SubscriptionBuiltinTopicData, 2017
- period
 - DDS_DeadlineQosPolicy, 1685
 - DDS_FlowControllerTokenBucketProperty_t, 1802
- periodic_settings
 - DDS_MonitoringDistributionSettings, 1840
- plugin_bitmask
 - DDS_EndpointTrustProtectionInfo, 1788
 - DDS_ParticipantTrustProtectionInfo, 1880
- plugin_data
 - DDS_TypeSupportQosPolicy, 2082
- policies
 - DDS_OfferedIncompatibleQosStatus, 1863
 - DDS_RequestedIncompatibleQosStatus, 1949
- policy_id
 - DDS_QosPolicyCount, 1925
- polling_period
 - DDS_MonitoringPeriodicDistributionSettings, 1853
- port
 - DDS_Locator_t, 1827
- port_base
 - DDS_RtpsWellKnownPorts_t, 1980
- port_offset
 - NDDS_Transport_UDPv4_Property_t, 2143
 - NDDS_Transport_UDPv4_WAN_Property_t, 2153
 - NDDS_Transport_UDPv6_Property_t, 2163
- PRESENTATION, 1304
 - DDS_GROUP_PRESENTATION_QOS, 1305
 - DDS_HIGHEST_OFFERED_PRESENTATION_QOS, 1305
 - DDS_INSTANCE_PRESENTATION_QOS, 1305
 - DDS_PRESENTATION_QOS_POLICY_NAME, 1305
 - DDS_PresentationQosPolicyAccessScopeKind, 1304
 - DDS_TOPIC_PRESENTATION_QOS, 1305
- presentation
 - DDS_PublicationBuiltinTopicData, 1909
 - DDS_PublisherQos, 1919
 - DDS_SubscriberQos, 2011
 - DDS_SubscriptionBuiltinTopicData, 2016
- pretty_print
 - DDS_PrintFormatProperty, 1895
- prevent_type_widening
 - DDS_TypeConsistencyEnforcementQosPolicy, 2078
- print_complete_type
 - DDS_TypeCodePrintFormatProperty, 2075
- print_format
 - DDS_LoggingQosPolicy, 1832
- print_kind
 - DDS_TypeCodePrintFormatProperty, 2074
- print_ordinals
 - DDS_TypeCodePrintFormatProperty, 2073
- print_private
 - DDS_QosPrintFormat, 1927
- priority
 - DDS_ChannelSettings_t, 1565
 - DDS_PublishModeQosPolicy, 1923
 - DDS_ThreadSettings_t, 2030
 - DDS_WriteParams_t, 2103
- product_version
 - DDS_ParticipantBuiltinTopicData, 1873
 - DDS_PublicationBuiltinTopicData, 1912
 - DDS_SubscriptionBuiltinTopicData, 2020
- PROFILE, 1305

- DDS_PROFILE_QOS_POLICY_NAME, 1306
- profile
 - DDS_DomainParticipantFactoryQos, 1722
- Programming How-To's, 1018
- propagate
 - DDS_Property_t, 1901
- propagate_app_ack_with_no_response
 - DDS_DataWriterProtocolQosPolicy, 1653
- propagate_dispose_of_unregistered_instances
 - DDS_DataReaderProtocolQosPolicy, 1598
- propagate_unregister_of_disposed_instances
 - DDS_DataReaderProtocolQosPolicy, 1599
- properties_bitmap
 - NDDS_Transport_Property_t, 2124
- PROPERTY, 1306
 - DDS_PROPERTY_QOS_IMMUTABLE, 1310
 - DDS_PROPERTY_QOS_MUTABLE, 1310
 - DDS_PROPERTY_QOS_MUTABLE_UNTIL_ENABLE, 1310
 - DDS_PROPERTY_QOS_POLICY_NAME, 1317
 - DDS_PropertyQosPolicy, 1308
 - DDS_PropertyQosPolicyHelper_add_pointer_property, 1312
 - DDS_PropertyQosPolicyHelper_add_property, 1311
 - DDS_PropertyQosPolicyHelper_assert_pointer_property, 1312
 - DDS_PropertyQosPolicyHelper_assert_property, 1310
 - DDS_PropertyQosPolicyHelper_get_number_of_properties, 1310
 - DDS_PropertyQosPolicyHelper_get_properties, 1315
 - DDS_PropertyQosPolicyHelper_get_properties_into_policy, 1315
 - DDS_PropertyQosPolicyHelper_get_property_mutability, 1316
 - DDS_PropertyQosPolicyHelper_lookup_property, 1313
 - DDS_PropertyQosPolicyHelper_lookup_property_with_prefix, 1314
 - DDS_PropertyQosPolicyHelper_remove_property, 1314
 - DDS_PropertyQosPolicyMutability, 1309
- property
 - DDS_DataReaderQos, 1616
 - DDS_DataWriterQos, 1672
 - DDS_DomainParticipantQos, 1730
 - DDS_ParticipantBuiltinTopicData, 1871
 - DDS_PublicationBuiltinTopicData, 1911
 - DDS_SubscriptionBuiltinTopicData, 2018
- protocol
 - DDS_DataReaderQos, 1616
 - DDS_DataWriterQos, 1671
- protocol_overhead_max
 - NDDS_Transport_UDPv4_Property_t, 2142
 - NDDS_Transport_UDPv4_WAN_Property_t, 2150
 - NDDS_Transport_UDPv6_Property_t, 2161
- public_address
 - NDDS_Transport_UDPv4_Property_t, 2143
 - NDDS_Transport_UDPv4_WAN_Property_t, 2151
 - NDDS_Transport_UDPv6_Property_t, 2162
- public_port
 - NDDS_Transport_UDP_WAN_CommPortsMappingInfo, 2133
- Publication Built-in Topics, 1094
 - DDS_PUBLICATION_TOPIC_NAME, 1096
 - DDS_PublicationBuiltinTopicData, 1095
 - DDS_PublicationBuiltinTopicDataDataReader, 1095
- Publication Example, 973
- Publication Module, 438
- publication_handle
 - DDS_SampleInfo, 1990
- publication_name
 - DDS_DataWriterQos, 1672
 - DDS_PublicationBuiltinTopicData, 1912
- publication_period
 - DDS_MonitoringEventDistributionSettings, 1843
 - DDS_MonitoringLoggingDistributionSettings, 1846
- DDS_TopicQueryDispatchQosPolicy, 2048
- publication_reader
 - DDS_DiscoveryConfigQosPolicy, 1696
- publication_reader_resource_limits
 - DDS_DiscoveryConfigQosPolicy, 1697
- publication_sequence_number
 - DDS_SampleInfo, 1992
- publication_writer
 - DDS_DiscoveryConfigQosPolicy, 1698
- publication_writer_data_lifecycle
 - DDS_DiscoveryConfigQosPolicy, 1699
- publication_writer_publish_mode
 - DDS_DiscoveryConfigQosPolicy, 1703
- PUBLISH_MODE, 1317
 - DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS, 1319
 - DDS_PUBLICATION_PRIORITY_AUTOMATIC, 1318
 - DDS_PUBLICATION_PRIORITY_UNDEFINED, 1318
 - DDS_PUBLISHMODE_QOS_POLICY_NAME, 1319
 - DDS_PublishModeQosPolicyKind, 1318
 - DDS_SYNCHRONOUS_PUBLISH_MODE_QOS, 1319
- publish_mode
 - DDS_DataWriterQos, 1671
- publisher
 - RTI_Connext_ReplierParams, 2172
 - RTI_Connext_RequesterParams, 2176
 - RTI_Connext_SimpleReplierParams, 2180
- Publisher Use Cases, 981

- publisher_group_data_max_length
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1746
- publisher_key
 - DDS_PublicationBuiltinTopicData, 1911
- publisher_name
 - DDS_PublisherQos, 1919
- publisher_qos_profile_name
 - DDS_MonitoringDistributionSettings, 1839
- Publishers, 439
 - DDS_DATAWRITER_QOS_DEFAULT, 470
 - DDS_DATAWRITER_QOS_PRINT_ALL, 469
 - DDS_DATAWRITER_QOS_USE_TOPIC_QOS, 470
 - DDS_Publisher, 443
 - DDS_Publisher_as_entity, 448
 - DDS_Publisher_begin_coherent_changes, 459
 - DDS_Publisher_copy_from_topic_qos, 462
 - DDS_Publisher_create_datawriter, 453
 - DDS_Publisher_create_datawriter_with_profile, 455
 - DDS_Publisher_delete_contained_entities, 461
 - DDS_Publisher_delete_datawriter, 456
 - DDS_Publisher_end_coherent_changes, 460
 - DDS_Publisher_get_all_datawriters, 460
 - DDS_Publisher_get_default_datawriter_qos, 449
 - DDS_Publisher_get_default_library, 464
 - DDS_Publisher_get_default_profile, 465
 - DDS_Publisher_get_default_profile_library, 465
 - DDS_Publisher_get_listener, 466
 - DDS_Publisher_get_listenerX, 467
 - DDS_Publisher_get_participant, 461
 - DDS_Publisher_get_qos, 464
 - DDS_Publisher_lookup_datawriter, 457
 - DDS_Publisher_lookup_datawriter_by_name, 469
 - DDS_Publisher_resume_publications, 458
 - DDS_Publisher_set_default_datawriter_qos, 450
 - DDS_Publisher_set_default_datawriter_qos_with_profile, 450
 - DDS_Publisher_set_default_library, 452
 - DDS_Publisher_set_default_profile, 451
 - DDS_Publisher_set_listener, 466
 - DDS_Publisher_set_qos, 462
 - DDS_Publisher_set_qos_with_profile, 463
 - DDS_Publisher_suspend_publications, 457
 - DDS_Publisher_wait_for_acknowledgments, 467
 - DDS_Publisher_wait_for_asynchronous_publishing, 468
 - DDS_PublisherListener_INITIALIZER, 442
 - DDS_PublisherQos_copy, 448
 - DDS_PublisherQos_equals, 444
 - DDS_PublisherQos_finalize, 447
 - DDS_PublisherQos_initialize, 446
 - DDS_PublisherQos_INITIALIZER, 442
 - DDS_PublisherQos_print, 444
 - DDS_PublisherQos_to_string, 445
 - DDS_PublisherQos_to_string_w_params, 446
 - pulled_fragment_bytes
 - DDS_DataWriterProtocolStatus, 1665
 - pulled_fragment_count
 - DDS_DataWriterProtocolStatus, 1665
 - pulled_sample_bytes
 - DDS_DataWriterProtocolStatus, 1659
 - pulled_sample_bytes_change
 - DDS_DataWriterProtocolStatus, 1660
 - pulled_sample_count
 - DDS_DataWriterProtocolStatus, 1659
 - pulled_sample_count_change
 - DDS_DataWriterProtocolStatus, 1659
 - push_on_write
 - DDS_DataWriterProtocolQosPolicy, 1650
 - pushed_fragment_bytes
 - DDS_DataWriterProtocolStatus, 1664
 - pushed_fragment_count
 - DDS_DataWriterProtocolStatus, 1664
 - pushed_sample_bytes
 - DDS_DataWriterProtocolStatus, 1657
 - pushed_sample_bytes_change
 - DDS_DataWriterProtocolStatus, 1657
 - pushed_sample_count
 - DDS_DataWriterProtocolStatus, 1657
 - pushed_sample_count_change
 - DDS_DataWriterProtocolStatus, 1657
 - QoS, 1238
 - QoS by Category, 1249
 - QoS Mutability, 914
 - QoS Profile Management, 893
 - qos_library_name
 - RTI_Connext_ReplierParams, 2170
 - RTI_Connext_RequesterParams, 2175
 - RTI_Connext_SimpleReplierParams, 2179
 - qos_profile_name
 - RTI_Connext_ReplierParams, 2171
 - RTI_Connext_RequesterParams, 2175
 - RTI_Connext_SimpleReplierParams, 2179
 - Queries and Filters Syntax, 738
 - Query Conditions, 698
 - DDS_QueryCondition, 699
 - DDS_QueryCondition_as_readcondition, 699
 - DDS_QueryCondition_get_query_expression, 699
 - DDS_QueryCondition_get_query_parameters, 700
 - DDS_QueryCondition_set_query_parameters, 700
 - query_condition_allocation
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1740
 - query_expression
 - DDS_QueryConditionParams, 1929
 - query_parameters
 - DDS_QueryConditionParams, 1929

- quorum_count
 - DDS_EndpointGroup_t, 1785
- rank
 - NDDS_Transport_Interface_t, 2121
- reachability_lease_duration
 - DDS_ParticipantBuiltinTopicData, 1874
- Read Conditions, 694
 - DDS_ReadCondition, 694
 - DDS_ReadCondition_as_condition, 695
 - DDS_ReadCondition_get_datareader, 696
 - DDS_ReadCondition_get_instance_state_mask, 696
 - DDS_ReadCondition_get_sample_state_mask, 695
 - DDS_ReadCondition_get_stream_kind_mask, 696
 - DDS_ReadCondition_get_view_state_mask, 695
- read_condition_allocation
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1739
- reader_checkpoint_frequency
 - DDS_PersistentStorageSettings, 1889
- READER_DATA_LIFECYCLE, 1320
 - DDS_READERDATA_LIFECYCLE_QOS_POLICY_NAME, 1320
- reader_data_lifecycle
 - DDS_DataReaderQos, 1615
- reader_data_tag_list_max_length
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1757
- reader_data_tag_string_max_length
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1758
- reader_property_list_max_length
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1754
- reader_property_string_max_length
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1754
- reader_resource_limits
 - DDS_DataReaderQos, 1615
- reader_user_data_max_length
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1747
- Real-Time WAN Transport, 1051
 - NDDS_Transport_UDPv4_WAN_create, 1056
 - NDDS_Transport_UDPv4_WAN_create_from_properties_with_prefix, 1057
 - NDDS_TRANSPORT_UDPv4_WAN_MESSAGE_SIZE_MAX_DEFAULT, 1055
 - NDDS_Transport_UDPv4_WAN_new, 1056
 - NDDS_TRANSPORT_UDPv4_WAN_PROPERTY_DEFAULT, 1055
- reassembled_sample_count
 - DDS_DataReaderProtocolStatus, 1610
- receive_address
 - DDS_TransportMulticastSettings_t, 2061
- receive_buffer_size
 - NDDS_Transport_Shmem_Property_t, 2130
- receive_port
 - DDS_TransportMulticastSettings_t, 2061
 - DDS_TransportUnicastSettings_t, 2068
- receive_window_size
 - DDS_RtpsReliableReaderProtocol_t, 1957
- received_ack_bytes
 - DDS_DataWriterProtocolStatus, 1660
- received_ack_bytes_change
 - DDS_DataWriterProtocolStatus, 1660
- received_ack_count
 - DDS_DataWriterProtocolStatus, 1660
- received_ack_count_change
 - DDS_DataWriterProtocolStatus, 1660
- received_fragment_count
 - DDS_DataReaderProtocolStatus, 1610
- received_gap_bytes
 - DDS_DataReaderProtocolStatus, 1608
- received_gap_bytes_change
 - DDS_DataReaderProtocolStatus, 1608
- received_gap_count
 - DDS_DataReaderProtocolStatus, 1607
- received_gap_count_change
 - DDS_DataReaderProtocolStatus, 1608
- received_heartbeat_bytes
 - DDS_DataReaderProtocolStatus, 1606
- received_heartbeat_bytes_change
 - DDS_DataReaderProtocolStatus, 1606
- received_heartbeat_count
 - DDS_DataReaderProtocolStatus, 1605
- received_heartbeat_count_change
 - DDS_DataReaderProtocolStatus, 1605
- received_message_count_max
 - NDDS_Transport_Shmem_Property_t, 2130
- received_nack_bytes
 - DDS_DataWriterProtocolStatus, 1661
- received_nack_bytes_change
 - DDS_DataWriterProtocolStatus, 1661
- received_nack_count
 - DDS_DataWriterProtocolStatus, 1661
- received_nack_count_change
 - DDS_DataWriterProtocolStatus, 1661
- received_nack_fragment_bytes
 - DDS_DataWriterProtocolStatus, 1665
- received_nack_fragment_count
 - DDS_DataWriterProtocolStatus, 1665
- received_sample_bytes
 - DDS_DataReaderProtocolStatus, 1603
- received_sample_bytes_change
 - DDS_DataReaderProtocolStatus, 1603
- received_sample_count
 - DDS_DataReaderProtocolStatus, 1602

- received_sample_count_change
 - DDS_DataReaderProtocolStatus, 1603
- RECEIVER_POOL, 1320
 - DDS_LENGTH_AUTO, 1321
 - DDS_RECEIVERPOOL_QOS_POLICY_NAME, 1321
- receiver_pool
 - DDS_DomainParticipantQos, 1729
- reception_sequence_number
 - DDS_SampleInfo, 1993
- reception_timestamp
 - DDS_SampleInfo, 1992
- recv_socket_buffer_size
 - NDDS_Transport_UDPv4_Property_t, 2136
 - NDDS_Transport_UDPv4_WAN_Property_t, 2146
 - NDDS_Transport_UDPv6_Property_t, 2156
- rejected_sample_count
 - DDS_DataReaderProtocolStatus, 1608
 - DDS_DataWriterProtocolStatus, 1662
- rejected_sample_count_change
 - DDS_DataReaderProtocolStatus, 1608
 - DDS_DataWriterProtocolStatus, 1662
- related_datareader_key
 - DDS_PublicationBuiltinTopicData, 1913
- related_datawriter_key
 - DDS_SubscriptionBuiltinTopicData, 2021
- related_original_publication_virtual_guid
 - DDS_SampleInfo, 1993
- related_original_publication_virtual_sequence_number
 - DDS_SampleInfo, 1994
- related_reader_guid
 - DDS_FilterSampleInfo, 1797
 - DDS_WriteParams_t, 2105
- related_sample_identity
 - DDS_FilterSampleInfo, 1796
 - DDS_WriteParams_t, 2102
- related_source_guid
 - DDS_FilterSampleInfo, 1797
 - DDS_SampleInfo, 1994
 - DDS_WriteParams_t, 2105
- related_subscription_guid
 - DDS_SampleInfo, 1995
- related_topic_name
 - DDS_ContentFilterProperty_t, 1579
- release
 - DDS_ProductVersion_t, 1897
 - NDDS_Config_LibraryVersion_t, 2115
- RELIABILITY, 1321
 - DDS_APPLICATION_AUTO_ACKNOWLEDGMENT_MODE, 1323
 - DDS_APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE, 1324
 - DDS_BEST_EFFORT_RELIABILITY_QOS, 1323
 - DDS_InstanceStateConsistencyKind, 1324
 - DDS_NO_RECOVER_INSTANCE_STATE_CONSISTENCY, 1324
 - DDS_PROTOCOL_ACKNOWLEDGMENT_MODE, 1323
 - DDS_RECOVER_INSTANCE_STATE_CONSISTENCY, 1324
 - DDS_RELIABILITY_QOS_POLICY_NAME, 1325
 - DDS_ReliabilityQosPolicyAcknowledgmentModeKind, 1323
 - DDS_ReliabilityQosPolicyKind, 1322
 - DDS_RELIABLE_RELIABILITY_QOS, 1323
- reliability
 - DDS_DataReaderQos, 1614
 - DDS_DataWriterQos, 1669
 - DDS_PublicationBuiltinTopicData, 1908
 - DDS_SubscriptionBuiltinTopicData, 2015
 - DDS_TopicBuiltinTopicData, 2037
 - DDS_TopicQos, 2044
- Reliability QoS, 895
- remote_participant_allocation
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1737
- remote_participant_hash_buckets
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1743
- remote_participant_purge_kind
 - DDS_DiscoveryConfigQosPolicy, 1693
- remote_reader_allocation
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1737
- remote_reader_hash_buckets
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1743
- remote_topic_query_allocation
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1756
- remote_topic_query_hash_buckets
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1756
- remote_writer_allocation
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1737
- remote_writer_hash_buckets
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1742
- replace_auto
 - DDS_WriteParams_t, 2102
- replace_empty_instances
 - DDS_DataWriterResourceLimitsQosPolicy, 1678
- replaced_dropped_sample_count
 - DDS_DataReaderCacheStatus, 1589
- replaced_unacknowledged_sample_count
 - DDS_ReliableWriterCacheChangedStatus, 1945
- Replier, 761

- FooBarReplier, 766
- FooBarReplier_create, 769
- FooBarReplier_create_w_params, 769
- FooBarReplier_get_reply_datawriter, 773
- FooBarReplier_get_request_datareader, 772
- FooBarReplier_read_request, 771
- FooBarReplier_read_requests, 771
- FooBarReplier_receive_request, 770
- FooBarReplier_receive_requests, 770
- FooBarReplier_return_loan, 773
- FooBarReplier_send_reply, 772
- FooBarReplier_take_request, 770
- FooBarReplier_take_requests, 771
- FooBarSimpleReplier, 767
- FooBarSimpleReplier_create, 773
- FooBarSimpleReplier_create_w_params, 774
- FooBarSimpleReplier_delete, 774
- on_request_available, 774, 775
- return_loan, 775
- RTI_CONNEXT_REPLIER_DECL, 764
- RTI_Connext_Replier_delete, 768
- RTI_Connext_Replier_wait_for_requests, 768
- RTI_Connext_ReplierListener_INITIALIZER, 764
- RTI_Connext_ReplierListener_OnRequestAvailableCallback, 765
- RTI_Connext_ReplierParams, 766
- RTI_Connext_ReplierParams_INITIALIZER, 764
- RTI_Connext_RequesterParams_INITIALIZER, 764
- RTI_CONNEXT_SIMPLEREPLIER_DECL, 765
- RTI_Connext_SimpleReplierListener_OnRequestAvailableCallback, 765
- RTI_Connext_SimpleReplierListener_OnReturnLoanCallback, 766
- RTI_Connext_SimpleReplierParams, 767
- RTI_Connext_SimpleReplierParams_INITIALIZER, 765
- reply_topic_name
 - RTI_Connext_ReplierParams, 2170
 - RTI_Connext_RequesterParams, 2175
 - RTI_Connext_SimpleReplierParams, 2179
- representation
 - DDS_DataReaderQos, 1615
 - DDS_DataWriterQos, 1670
 - DDS_PublicationBuiltinTopicData, 1910
 - DDS_SubscriptionBuiltinTopicData, 2017
 - DDS_TopicBuiltinTopicData, 2039
 - DDS_TopicQos, 2045
- Request-Reply Examples, 1002
- Request-Reply Pattern, 745
- request_body
 - DDS_ServiceRequest, 2002
- request_topic_name
 - RTI_Connext_ReplierParams, 2170
 - RTI_Connext_RequesterParams, 2174
- RTI_Connext_SimpleReplierParams, 2179
- Requester, 746
 - FooBarRequester, 748
 - FooBarRequester_create, 752
 - FooBarRequester_create_w_params, 752
 - FooBarRequester_get_reply_datareader, 760
 - FooBarRequester_get_request_datawriter, 759
 - FooBarRequester_read_replies, 758
 - FooBarRequester_read_replies_for_related_request, 759
 - FooBarRequester_read_reply, 758
 - FooBarRequester_read_reply_for_related_request, 759
 - FooBarRequester_receive_replies, 754
 - FooBarRequester_receive_reply, 754
 - FooBarRequester_return_loan, 760
 - FooBarRequester_send_request, 752
 - FooBarRequester_send_request_w_params, 753
 - FooBarRequester_take_replies, 756
 - FooBarRequester_take_replies_for_related_request, 757
 - FooBarRequester_take_reply, 755
 - FooBarRequester_take_reply_for_related_request, 757
 - RTI_CONNEXT_REQUESTER_DECL, 748
 - RTI_Connext_Requester_delete, 750
 - RTI_Connext_Requester_wait_for_replies, 750
 - RTI_Connext_Requester_wait_for_replies_for_related_request, 751
 - RTI_Connext_RequesterParams, 748
 - required_mask
 - DDS_EndpointTrustInterceptorAlgorithmInfo, 1787
 - DDS_TrustAlgorithmRequirements, 2069
 - required_matched_endpoint_groups
 - DDS_AvailabilityQosPolicy, 1551
 - RESOURCE_LIMITS, 1325
 - DDS_LENGTH_UNLIMITED, 1326
 - DDS_RESOURCELIMITS_QOS_POLICY_NAME, 1325
 - resource_limits
 - DDS_DataReaderQos, 1614
 - DDS_DataWriterQos, 1669
 - DDS_DomainParticipantFactoryQos, 1722
 - DDS_DomainParticipantQos, 1729
 - DDS_TopicBuiltinTopicData, 2038
 - DDS_TopicQos, 2045
 - resource_selection
 - DDS_MonitoringMetricSelection, 1849
 - response_data
 - DDS_AcknowledgmentInfo, 1536
 - restore
 - DDS_PersistentStorageSettings, 1887
 - Return Codes, 1220
 - DDS_RETCODE_ALREADY_DELETED, 1222

- DDS_RETCODE_BAD_PARAMETER, 1222
- DDS_RETCODE_ERROR, 1222
- DDS_RETCODE_ILLEGAL_OPERATION, 1222
- DDS_RETCODE_IMMUTABLE_POLICY, 1222
- DDS_RETCODE_INCONSISTENT_POLICY, 1222
- DDS_RETCODE_NO_DATA, 1222
- DDS_RETCODE_NOT_ALLOWED_BY_SECURITY, 1222
- DDS_RETCODE_NOT_ENABLED, 1222
- DDS_RETCODE_OK, 1222
- DDS_RETCODE_OUT_OF_RESOURCES, 1222
- DDS_RETCODE_PRECONDITION_NOT_MET, 1222
- DDS_RETCODE_TIMEOUT, 1222
- DDS_RETCODE_UNSUPPORTED, 1222
- DDS_ReturnCode_t, 1221
- return_loan
 - Replier, 775
- reuse_multicast_receive_resource
 - NDDS_Transport_UDPv4_Property_t, 2141
 - NDDS_Transport_UDPv6_Property_t, 2161
- revision
 - DDS_ProductVersion_t, 1898
- role_name
 - DDS_EndpointGroup_t, 1784
 - DDS_EntityNameQosPolicy, 1791
- round_trip_time
 - DDS_RtpsReliableReaderProtocol_t, 1957
- RTI_Connext DDS API Reference, 1013
- RTI_Connext_Messaging_get_api_build_number_string
 - Utilities, 777
- RTI_Connext_Messaging_get_api_version
 - Utilities, 776
- RTI_Connext_Messaging_Library_get_api_version_string
 - Utilities, 777
- RTI_Connext_Messaging_LibraryVersion
 - Utilities, 776
- RTI_Connext_Replier, 2167
- RTI_CONNEXT_REPLIER_DECL
 - Replier, 764
- RTI_Connext_Replier_delete
 - Replier, 768
- RTI_Connext_Replier_wait_for_requests
 - Replier, 768
- RTI_Connext_ReplierListener, 2168
 - user_data, 2168
- RTI_Connext_ReplierListener_INITIALIZER
 - Replier, 764
- RTI_Connext_ReplierListener_OnRequestAvailableCallback
 - Replier, 765
- RTI_Connext_ReplierParams, 2169
 - datareader_qos, 2171
 - datawriter_qos, 2171
 - listener, 2172
 - participant, 2170
 - publisher, 2172
 - qos_library_name, 2170
 - qos_profile_name, 2171
 - Replier, 766
 - reply_topic_name, 2170
 - request_topic_name, 2170
 - service_name, 2170
 - subscriber, 2172
- RTI_Connext_ReplierParams_INITIALIZER
 - Replier, 764
- RTI_Connext_Requester, 2173
- RTI_CONNEXT_REQUESTER_DECL
 - Requester, 748
- RTI_Connext_Requester_delete
 - Requester, 750
- RTI_Connext_Requester_wait_for_replies
 - Requester, 750
- RTI_Connext_Requester_wait_for_replies_for_related_request
 - Requester, 751
- RTI_Connext_RequesterParams, 2173
 - datareader_qos, 2176
 - datawriter_qos, 2175
 - participant, 2174
 - publisher, 2176
 - qos_library_name, 2175
 - qos_profile_name, 2175
 - reply_topic_name, 2175
 - request_topic_name, 2174
 - Requester, 748
 - service_name, 2174
 - subscriber, 2176
- RTI_Connext_RequesterParams_INITIALIZER
 - Replier, 764
- RTI_CONNEXT_SIMPLEREPLIER_DECL
 - Replier, 765
- RTI_Connext_SimpleReplierListener, 2176
 - user_data, 2177
- RTI_Connext_SimpleReplierListener_OnRequestAvailableCallback
 - Replier, 765
- RTI_Connext_SimpleReplierListener_OnReturnLoanCallback
 - Replier, 766
- RTI_Connext_SimpleReplierParams, 2177
 - datareader_qos, 2180
 - datawriter_qos, 2180
 - participant, 2178
 - publisher, 2180
 - qos_library_name, 2179
 - qos_profile_name, 2179
 - Replier, 767
 - reply_topic_name, 2179
 - request_topic_name, 2179
 - service_name, 2178
 - simple_listener, 2181

- subscriber, 2180
- RTI_Connext_SimpleReplierParams_INITIALIZER
 - Replier, 765
- RTI_MONITORING_EVENT_TOPIC_NAME
 - Monitoring Library 2.0, 1436
- RTI_Monitoring_initialize
 - Monitoring Library 2.0, 1435
- RTI_MONITORING_LOGGING_TOPIC_NAME
 - Monitoring Library 2.0, 1436
- RTI_MONITORING_PERIODIC_TOPIC_NAME
 - Monitoring Library 2.0, 1436
- rtps_app_id
 - DDS_WireProtocolQosPolicy, 2098
- rtps_auto_id_kind
 - DDS_WireProtocolQosPolicy, 2100
- rtps_host_id
 - DDS_WireProtocolQosPolicy, 2097
- rtps_instance_id
 - DDS_WireProtocolQosPolicy, 2098
- rtps_object_id
 - DDS_DataReaderProtocolQosPolicy, 1597
 - DDS_DataWriterProtocolQosPolicy, 1650
- rtps_port
 - NDDS_Transport_UDP_WAN_CommPortsMappingInfo, 2133
- rtps_protocol_version
 - DDS_ParticipantBuiltinTopicData, 1871
 - DDS_PublicationBuiltinTopicData, 1912
 - DDS_SubscriptionBuiltinTopicData, 2019
- rtps_reliable_reader
 - DDS_DataReaderProtocolQosPolicy, 1599
- rtps_reliable_writer
 - DDS_DataWriterProtocolQosPolicy, 1653
- rtps_reserved_port_mask
 - DDS_WireProtocolQosPolicy, 2099
- rtps_vendor_id
 - DDS_ParticipantBuiltinTopicData, 1871
 - DDS_PublicationBuiltinTopicData, 1912
 - DDS_SubscriptionBuiltinTopicData, 2019
- rtps_well_known_ports
 - DDS_WireProtocolQosPolicy, 2099
- Sample Flags, 1386
 - DDS_DISCOVERY_SERVICE_SAMPLE, 1388
 - DDS_INCOMPLETE_SNAPSHOT_TOPIC_QUERY, 1388
 - DDS_INTERMEDIATE_REPLY_SEQUENCE_SAMPLE, 1388
 - DDS_INTERMEDIATE_TOPIC_QUERY_SAMPLE, 1388
 - DDS_LAST_SHARED_READER_QUEUE_SAMPLE, 1388
 - DDS_REDELIVERED_SAMPLE, 1388
 - DDS_REPLICATE_SAMPLE, 1388
 - DDS_SampleFlag, 1387
 - DDS_SampleFlagBits, 1387, 1388
 - DDS_WRITER_REMOVED_BATCH_SAMPLE, 1388
- Sample Memory Management QoS, 902
- Sample Ordering, 905
- Sample States, 709
 - DDS_ANY_SAMPLE_STATE, 711
 - DDS_NOT_READ_SAMPLE_STATE, 711
 - DDS_READ_SAMPLE_STATE, 711
 - DDS_SampleStateKind, 710
 - DDS_SampleStateMask, 710
- sample_count
 - DDS_DataReaderCacheStatus, 1587
 - DDS_DataWriterCacheStatus, 1642
- sample_count_peak
 - DDS_DataReaderCacheStatus, 1587
 - DDS_DataWriterCacheStatus, 1642
- sample_identity
 - DDS_AcknowledgmentInfo, 1536
 - DDS_FilterSampleInfo, 1797
- sample_rank
 - DDS_SampleInfo, 1991
- sample_state
 - DDS_SampleInfo, 1988
- sample_states
 - DDS_ReadConditionParams, 1930
- SampleProcessor, 1503
 - DDS_SampleHandler_INITIALIZER, 1504
 - DDS_SampleProcessor, 1505
 - DDS_SampleProcessor_attach_reader, 1506
 - DDS_SampleProcessor_delete, 1509
 - DDS_SampleProcessor_detach_reader, 1507
 - DDS_SampleProcessor_get_datareaders, 1508
 - DDS_SampleProcessor_lookup_sample_handler, 1508
 - DDS_SampleProcessor_new, 1510
 - DDS_SampleProcessor_new_with_aws, 1510
- samples_per_app_ack
 - DDS_RtpsReliableReaderProtocol_t, 1959
- samples_per_period
 - DDS_TopicQueryDispatchQosPolicy, 2049
- samples_per_virtual_heartbeat
 - DDS_RtpsReliableWriterProtocol_t, 1965
- scheduling_policy
 - DDS_FlowControllerProperty_t, 1799
- scope
 - DDS_DestinationOrderQosPolicy, 1688
- sec
 - DDS_Duration_t, 1767
 - DDS_Time_t, 2031
- secure_volatile_reader
 - DDS_DiscoveryConfigQosPolicy, 1710
- secure_volatile_writer
 - DDS_DiscoveryConfigQosPolicy, 1709

- secure_volatile_writer_publish_mode
 - DDS_DiscoveryConfigQosPolicy, 1710
- Security, 906
- security_event_forwarding_level
 - DDS_MonitoringLoggingForwardingSettings, 1847
- send_blocking
 - NDDS_Transport_UDPv4_Property_t, 2139
 - NDDS_Transport_UDPv4_WAN_Property_t, 2148
 - NDDS_Transport_UDPv6_Property_t, 2159
- send_ping
 - NDDS_Transport_UDPv4_Property_t, 2141
 - NDDS_Transport_UDPv4_WAN_Property_t, 2149
 - NDDS_Transport_UDPv6_Property_t, 2160
- send_socket_buffer_size
 - NDDS_Transport_UDPv4_Property_t, 2135
 - NDDS_Transport_UDPv4_WAN_Property_t, 2145
 - NDDS_Transport_UDPv6_Property_t, 2155
- send_window_decrease_factor
 - DDS_RtpsReliableWriterProtocol_t, 1976
- send_window_increase_factor
 - DDS_RtpsReliableWriterProtocol_t, 1975
- send_window_size
 - DDS_DataWriterProtocolStatus, 1662
- send_window_update_period
 - DDS_RtpsReliableWriterProtocol_t, 1974
- Sending Large Data, 871
- sent_ack_bytes
 - DDS_DataReaderProtocolStatus, 1606
- sent_ack_bytes_change
 - DDS_DataReaderProtocolStatus, 1607
- sent_ack_count
 - DDS_DataReaderProtocolStatus, 1606
- sent_ack_count_change
 - DDS_DataReaderProtocolStatus, 1606
- sent_gap_bytes
 - DDS_DataWriterProtocolStatus, 1662
- sent_gap_bytes_change
 - DDS_DataWriterProtocolStatus, 1662
- sent_gap_count
 - DDS_DataWriterProtocolStatus, 1661
- sent_gap_count_change
 - DDS_DataWriterProtocolStatus, 1662
- sent_heartbeat_bytes
 - DDS_DataWriterProtocolStatus, 1659
- sent_heartbeat_bytes_change
 - DDS_DataWriterProtocolStatus, 1659
- sent_heartbeat_count
 - DDS_DataWriterProtocolStatus, 1658
- sent_heartbeat_count_change
 - DDS_DataWriterProtocolStatus, 1658
- sent_nack_bytes
 - DDS_DataReaderProtocolStatus, 1607
- sent_nack_bytes_change
 - DDS_DataReaderProtocolStatus, 1607
- sent_nack_count
 - DDS_DataReaderProtocolStatus, 1607
- sent_nack_count_change
 - DDS_DataReaderProtocolStatus, 1607
- sent_nack_fragment_bytes
 - DDS_DataReaderProtocolStatus, 1610
- sent_nack_fragment_count
 - DDS_DataReaderProtocolStatus, 1610
- Sequence Number Support, 1214
 - DDS_AUTO_SEQUENCE_NUMBER, 1218
 - DDS_SEQUENCE_NUMBER_MAX, 1218
 - DDS_SEQUENCE_NUMBER_UNKNOWN, 1218
 - DDS_SEQUENCE_NUMBER_ZERO, 1218
 - DDS_SequenceNumber_add, 1215
 - DDS_SequenceNumber_compare, 1216
 - DDS_SequenceNumber_minusminus, 1216
 - DDS_SequenceNumber_plusplus, 1216
 - DDS_SequenceNumber_subtract, 1215
 - DDS_SequenceNumber_t, 1215
- Sequence Support, 1511
 - DDS_SEQUENCE_INITIALIZER, 1513
 - FooSeq_copy, 1519
 - FooSeq_copy_no_alloc, 1518
 - FooSeq_ensure_length, 1516
 - FooSeq_finalize, 1525
 - FooSeq_from_array, 1519
 - FooSeq_get, 1517
 - FooSeq_get_contiguous_buffer, 1523
 - FooSeq_get_discontiguous_buffer, 1524
 - FooSeq_get_length, 1515
 - FooSeq_get_maximum, 1514
 - FooSeq_get_reference, 1517
 - FooSeq_has_ownership, 1524
 - FooSeq_initialize, 1513
 - FooSeq_loan_contiguous, 1521
 - FooSeq_loan_discontiguous, 1522
 - FooSeq_set_length, 1515
 - FooSeq_set_maximum, 1514
 - FooSeq_to_array, 1520
 - FooSeq_unloan, 1522
- sequence_number
 - WriteParams, 1391
- serialization
 - DDS_DynamicDataTypeProperty_t, 1782
- serialize_key_with_dispose
 - DDS_DataWriterProtocolQosPolicy, 1652
- serialized_type_object_dynamic_allocation_threshold
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1750
- SERVICE, 1326
 - DDS_ADMIN_CONSOLE_SERVICE_QOS, 1327
 - DDS_DATABASE_INTEGRATION_SERVICE_QOS, 1327
 - DDS_NO_SERVICE_QOS, 1327

- DDS_OBSERVABILITY_COLLECTOR_SERVICE_QOS, 1327
- DDS_PERSISTENCE_SERVICE_QOS, 1327
- DDS_QUEUEING_SERVICE_QOS, 1327
- DDS_RECORDING_SERVICE_QOS, 1327
- DDS_REPLAY_SERVICE_QOS, 1327
- DDS_ROUTING_SERVICE_QOS, 1327
- DDS_SERVICE_QOS_POLICY_NAME, 1328
- DDS_ServiceQosPolicyKind, 1327
- DDS_WEB_INTEGRATION_SERVICE_QOS, 1327
- service
 - DDS_DataReaderQos, 1617
 - DDS_DataWriterQos, 1672
 - DDS_DomainParticipantQos, 1730
 - DDS_PublicationBuiltinTopicData, 1911
 - DDS_SubscriptionBuiltinTopicData, 2019
- service_cleanup_delay
 - DDS_DurabilityServiceQosPolicy, 1765
- service_forwarding_level
 - DDS_MonitoringLoggingForwardingSettings, 1848
- service_id
 - DDS_ServiceRequest, 2001
 - DDS_ServiceRequestAcceptedStatus, 2004
- service_name
 - RTI_Connext_ReplierParams, 2170
 - RTI_Connext_RequesterParams, 2174
 - RTI_Connext_SimpleReplierParams, 2178
- service_request_reader
 - DDS_DiscoveryConfigQosPolicy, 1707
- service_request_writer
 - DDS_DiscoveryConfigQosPolicy, 1705
- service_request_writer_data_lifecycle
 - DDS_DiscoveryConfigQosPolicy, 1706
- service_request_writer_publish_mode
 - DDS_DiscoveryConfigQosPolicy, 1706
- ServiceRequest Built-in Topic, 1098
 - DDS_INSTANCE_STATE_SERVICE_REQUEST_ID, 1101
 - DDS_LOCATOR_REACHABILITY_SERVICE_REQUEST_ID, 1101
 - DDS_MONITORING_LIBRARY_COMMAND_SERVICE_REQUEST_ID, 1101
 - DDS_MONITORING_LIBRARY_REPLY_SERVICE_REQUEST_ID, 1101
 - DDS_SERVICE_REQUEST_TOPIC_NAME, 1101
 - DDS_ServiceRequest, 1100
 - DDS_ServiceRequestDataReader, 1100
 - DDS_TOPIC_QUERY_SERVICE_REQUEST_ID, 1101
 - DDS_UNKNOWN_SERVICE_REQUEST_ID, 1100
- Shared Memory Transport, 1035
 - NDDS_TRANSPORT_SHMEM_ADDRESS_BIT_COUNT, 1039
 - NDDS_Transport_Shmem_create, 1041
- NDDS_TRANSPORT_SHMEM_GATHER_SEND_BUFFER_COUNT, 1039
- NDDS_TRANSPORT_SHMEM_MAJOR_AFTER_BUG_14240_FIX, 1040
- NDDS_TRANSPORT_SHMEM_MESSAGE_SIZE_MAX_DEFAULT, 1039
- NDDS_Transport_Shmem_new, 1040
- NDDS_TRANSPORT_SHMEM_PROPERTIES_BITMAP_DEFAULT, 1039
- NDDS_TRANSPORT_SHMEM_PROPERTY_DEFAULT, 1040
- NDDS_TRANSPORT_SHMEM_RECEIVE_BUFFER_SIZE_DEFAULT, 1040
- NDDS_TRANSPORT_SHMEM_RECEIVED_MESSAGE_COUNT_MAX, 1040
- shared_secret
 - DDS_ParticipantTrustKeyEstablishmentAlgorithmInfo, 1879
- shmem_ref_settings
 - DDS_DataWriterTransferModeQosPolicy, 1683
- shmem_ref_transfer_mode_attached_segment_allocation
 - DDS_DataReaderResourceLimitsQosPolicy, 1635
- shmem_ref_transfer_mode_max_segments
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1758
- shutdown_cleanup_period
 - DDS_DatabaseQosPolicy, 1583
- shutdown_timeout
 - DDS_DatabaseQosPolicy, 1583
- signature
 - DDS_ParticipantTrustAlgorithmInfo, 1877
- simple_listener
 - RTI_Connext_SimpleReplierParams, 2181
- skip_deserialization
 - DDS_DynamicDataTypeSerializationProperty_t, 1783
- snapshot_content_format
 - NDDS_Utility_HeapMonitoringParams_t, 2164
- snapshot_output_format
 - NDDS_Utility_HeapMonitoringParams_t, 2164
- source_id
 - DDS_FilterSampleInfo, 1798
 - DDS_SampleInfo, 1994
 - DDS_WriteParams_t, 2104
- source_timestamp
 - DDS_SampleInfo, 1989
 - DDS_WriteParams_t, 2103
- source_timestamp_resolution
 - DDS_BatchQosPolicy, 1554
- source_timestamp_tolerance
 - DDS_DestinationOrderQosPolicy, 1688
- stack_size
 - DDS_ThreadSettings_t, 2030
- status

- NDDS_Transport_Interface_t, 2121
- Status Kinds, 1222
 - DDS_DATA_AVAILABLE_STATUS, 1230
 - DDS_DATA_ON_READERS_STATUS, 1230
 - DDS_DATA_READER_CACHE_STATUS, 1235
 - DDS_DATA_READER_PROTOCOL_STATUS, 1235
 - DDS_DATA_WRITER_APPLICATION_ACKNOWLEDGMENT_STATUS, 1233
 - DDS_DATA_WRITER_CACHE_STATUS, 1234
 - DDS_DATA_WRITER_EMPTY_VIRTUAL_WRITER_STATUS, 1232
 - DDS_DATA_WRITER_INSTANCE_REPLACED_STATUS, 1233
 - DDS_DATA_WRITER_PROTOCOL_STATUS, 1234
 - DDS_DATA_WRITER_SAMPLE_REMOVED_STATUS, 1235
 - DDS_INCONSISTENT_TOPIC_STATUS, 1228
 - DDS_INVALID_LOCAL_ADVANCE_NOTICE_STATUS, 1232
 - DDS_LIVELINESS_CHANGED_STATUS, 1231
 - DDS_LIVELINESS_LOST_STATUS, 1231
 - DDS_OFFERED_DEADLINE_MISSED_STATUS, 1228
 - DDS_OFFERED_INCOMPATIBLE_QOS_STATUS, 1229
 - DDS_PUBLICATION_MATCHED_STATUS, 1232
 - DDS_RELIABLE_READER_ACTIVITY_CHANGED_STATUS, 1234
 - DDS_RELIABLE_WRITER_CACHE_CHANGED_STATUS, 1234
 - DDS_REQUESTED_DEADLINE_MISSED_STATUS, 1229
 - DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS, 1229
 - DDS_SAMPLE_LOST_STATUS, 1230
 - DDS_SAMPLE_REJECTED_STATUS, 1230
 - DDS_SERVICE_REQUEST_ACCEPTED_STATUS, 1233
 - DDS_STATUS_MASK_ALL, 1226
 - DDS_STATUS_MASK_NONE, 1226
 - DDS_StatusKind, 1227
 - DDS_StatusMask, 1226
 - DDS_SUBSCRIPTION_MATCHED_STATUS, 1232
- storage_settings
 - DDS_DurabilityQosPolicy, 1763
- stored_size
 - DDS_DynamicDataInfo, 1776
- Stream Kinds, 715
 - DDS_LIVE_STREAM, 716
 - DDS_StreamKind, 716
 - DDS_StreamKindMask, 716
 - DDS_TOPIC_QUERY_STREAM, 716
- stream_kinds
 - DDS_ReadConditionParams, 1930
- String Built-in Type, 1111
 - DDS_StringDataReader, 1114
 - DDS_StringDataReader_as_datareader, 1121
 - DDS_StringDataReader_narrow, 1120
 - DDS_StringDataReader_read, 1121
 - DDS_StringDataReader_read_next_sample, 1122
 - DDS_StringDataReader_read_w_condition, 1122
 - DDS_StringDataReader_return_loan, 1123
 - DDS_StringDataReader_take, 1121
 - DDS_StringDataReader_take_next_sample, 1123
 - DDS_StringDataReader_take_w_condition, 1122
 - DDS_StringDataWriter, 1114
 - DDS_StringDataWriter_as_datawriter, 1118
 - DDS_StringDataWriter_create_data, 1119
 - DDS_StringDataWriter_delete_data, 1119
 - DDS_StringDataWriter_narrow, 1118
 - DDS_StringDataWriter_write, 1119
 - DDS_StringDataWriter_write_w_params, 1120
 - DDS_StringDataWriter_write_w_timestamp, 1120
 - DDS_StringTypeSupport_data_to_string, 1118
 - DDS_StringTypeSupport_deserialize_data_from_cdr_buffer, 1117
 - DDS_StringTypeSupport_get_type_name, 1116
 - DDS_StringTypeSupport_get_typecode, 1117
 - DDS_StringTypeSupport_print_data, 1116
 - DDS_StringTypeSupport_register_type, 1114
 - DDS_StringTypeSupport_serialize_data_to_cdr_buffer, 1117
 - DDS_StringTypeSupport_serialize_data_to_cdr_buffer_ex, 1117
 - DDS_StringTypeSupport_unregister_type, 1115
- String Support, 1526
 - DDS_String_alloc, 1528
 - DDS_String_dup, 1528
 - DDS_String_free, 1529
 - DDS_String_replace, 1529
 - DDS_Wstring_alloc, 1530
 - DDS_Wstring_copy, 1531
 - DDS_Wstring_copy_and_widen, 1531
 - DDS_Wstring_dup, 1532
 - DDS_Wstring_dup_and_widen, 1532
 - DDS_Wstring_free, 1532
 - DDS_Wstring_length, 1530
- string_profile
 - DDS_ProfileQosPolicy, 1899
- subscriber
 - RTI_Connext_ReplierParams, 2172
 - RTI_Connext_RequesterParams, 2176
 - RTI_Connext_SimpleReplierParams, 2180
- Subscriber Use Cases, 983
- subscriber_group_data_max_length
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1747
- subscriber_key

- DDS_SubscriptionBuiltinTopicData, 2018
- subscriber_name
 - DDS_SubscriberQos, 2011
- Subscribers, 568
 - DDS_DATAREADER_QOS_DEFAULT, 600
 - DDS_DATAREADER_QOS_PRINT_ALL, 600
 - DDS_DATAREADER_QOS_USE_TOPIC_QOS, 601
 - DDS_Subscriber, 572
 - DDS_Subscriber_as_entity, 578
 - DDS_Subscriber_begin_access, 587
 - DDS_Subscriber_copy_from_topic_qos, 591
 - DDS_Subscriber_create_datareader, 581
 - DDS_Subscriber_create_datareader_with_profile, 583
 - DDS_Subscriber_delete_contained_entities, 585
 - DDS_Subscriber_delete_datareader, 585
 - DDS_Subscriber_end_access, 588
 - DDS_Subscriber_get_all_datareaders, 589
 - DDS_Subscriber_get_datareaders, 588
 - DDS_Subscriber_get_default_datareader_qos, 579
 - DDS_Subscriber_get_default_library, 596
 - DDS_Subscriber_get_default_profile, 594
 - DDS_Subscriber_get_default_profile_library, 595
 - DDS_Subscriber_get_listener, 597
 - DDS_Subscriber_get_listenerX, 598
 - DDS_Subscriber_get_participant, 590
 - DDS_Subscriber_get_qos, 593
 - DDS_Subscriber_lookup_datareader, 586
 - DDS_Subscriber_lookup_datareader_by_name, 598
 - DDS_Subscriber_notify_datareaders, 590
 - DDS_Subscriber_set_default_datareader_qos, 580
 - DDS_Subscriber_set_default_datareader_qos_with_profile, 580
 - DDS_Subscriber_set_default_library, 595
 - DDS_Subscriber_set_default_profile, 594
 - DDS_Subscriber_set_listener, 597
 - DDS_Subscriber_set_qos, 592
 - DDS_Subscriber_set_qos_with_profile, 592
 - DDS_SubscriberListener_DataOnReadersCallback, 573
 - DDS_SubscriberListener_INITIALIZER, 572
 - DDS_SubscriberQos_copy, 578
 - DDS_SubscriberQos_equals, 573
 - DDS_SubscriberQos_finalize, 577
 - DDS_SubscriberQos_initialize, 577
 - DDS_SubscriberQos_INITIALIZER, 571
 - DDS_SubscriberQos_print, 575
 - DDS_SubscriberQos_to_string, 575
 - DDS_SubscriberQos_to_string_w_params, 576
- Subscription Built-in Topics, 1096
 - DDS_SUBSCRIPTION_TOPIC_NAME, 1098
 - DDS_SubscriptionBuiltinTopicData, 1097
 - DDS_SubscriptionBuiltinTopicDataDataReader, 1097
- Subscription Example, 974
- Subscription Module, 567
- subscription_handle
 - DDS_AcknowledgmentInfo, 1535
- subscription_name
 - DDS_DataReaderQos, 1617
 - DDS_SubscriptionBuiltinTopicData, 2020
- subscription_reader
 - DDS_DiscoveryConfigQosPolicy, 1697
- subscription_reader_resource_limits
 - DDS_DiscoveryConfigQosPolicy, 1698
- subscription_writer
 - DDS_DiscoveryConfigQosPolicy, 1700
- subscription_writer_data_lifecycle
 - DDS_DiscoveryConfigQosPolicy, 1700
- subscription_writer_publish_mode
 - DDS_DiscoveryConfigQosPolicy, 1704
- supported_mask
 - DDS_EndpointTrustInterceptorAlgorithmInfo, 1787
 - DDS_ParticipantTrustInterceptorAlgorithmInfo, 1878
 - DDS_TrustAlgorithmRequirements, 2069
- synchronization_kind
 - DDS_PersistentStorageSettings, 1886
- System Properties, 970
- SYSTEM_RESOURCE_LIMITS, 1328
 - DDS_SYSTEMRESOURCELIMITS_QOS_POLICY_NAME, 1328
- tags
 - DDS_DataTags, 1641
- telemetry_data
 - DDS_MonitoringQosPolicy, 1855
- text
 - NDDS_Config_LogMessage, 2118
- thread
 - DDS_AsynchronousPublisherQosPolicy, 1540
 - DDS_DatabaseQosPolicy, 1582
 - DDS_EventQosPolicy, 1793
 - DDS_MonitoringEventDistributionSettings, 1842
 - DDS_MonitoringLoggingDistributionSettings, 1846
 - DDS_MonitoringPeriodicDistributionSettings, 1852
 - DDS_ReceiverPoolQosPolicy, 1935
- Thread Settings, 1235
 - DDS_THREAD_SETTINGS_CANCEL_ASYNCHRONOUS, 1237
 - DDS_THREAD_SETTINGS_CPU_NO_ROTATION, 1238
 - DDS_THREAD_SETTINGS_CPU_RR_ROTATION, 1238
 - DDS_THREAD_SETTINGS_FLOATING_POINT, 1237
 - DDS_THREAD_SETTINGS_KIND_MASK_DEFAULT, 1236

- DDS_THREAD_SETTINGS_PRIORITY_ENFORCE, 1237
- DDS_THREAD_SETTINGS_REALTIME_PRIORITY, 1237
- DDS_THREAD_SETTINGS_STDIO, 1237
- DDS_ThreadSettingsCpuRotationKind, 1237
- DDS_ThreadSettingsKind, 1237
- DDS_ThreadSettingsKindMask, 1236
- thread_name_prefix
 - DDS_AsyncWaitSetProperty_t, 1546
 - NDDS_Transport_Property_t, 2128
- thread_pool_size
 - DDS_AsyncWaitSetProperty_t, 1545
- thread_safe_write
 - DDS_BatchQosPolicy, 1555
- thread_settings
 - DDS_AsyncWaitSetProperty_t, 1546
- Throughput, 908
- Time Support, 1205
 - DDS_DURATION_AUTO, 1209
 - DDS_DURATION_AUTO_NSEC, 1209
 - DDS_DURATION_AUTO_SEC, 1209
 - DDS_DURATION_INFINITE, 1208
 - DDS_DURATION_INFINITE_NSEC, 1208
 - DDS_DURATION_INFINITE_SEC, 1208
 - DDS_Duration_is_auto, 1207
 - DDS_Duration_is_infinite, 1207
 - DDS_Duration_is_zero, 1207
 - DDS_DURATION_ZERO, 1209
 - DDS_DURATION_ZERO_NSEC, 1209
 - DDS_DURATION_ZERO_SEC, 1209
 - DDS_TIME_INVALID, 1208
 - DDS_TIME_INVALID_NSEC, 1208
 - DDS_TIME_INVALID_SEC, 1208
 - DDS_Time_is_invalid, 1206
 - DDS_Time_is_zero, 1206
 - DDS_TIME_MAX, 1207
 - DDS_TIME_ZERO, 1206
- TIME_BASED_FILTER, 1329
 - DDS_TIMEBASEDFILTER_QOS_POLICY_NAME, 1329
- time_based_filter
 - DDS_DataReaderQos, 1615
 - DDS_SubscriptionBuiltinTopicData, 2016
- time_based_filter_dropped_sample_count
 - DDS_DataReaderCacheStatus, 1588
- timestamp
 - NDDS_Config_LogMessage, 2119
- token_bucket
 - DDS_FlowControllerProperty_t, 1800
- tokens_added_per_period
 - DDS_FlowControllerTokenBucketProperty_t, 1801
- tokens_leaked_per_period
 - DDS_FlowControllerTokenBucketProperty_t, 1802
- tolerance_source_timestamp_dropped_sample_count
 - DDS_DataReaderCacheStatus, 1588
- Topic Built-in Topics, 1093
 - DDS_TOPIC_TOPIC_NAME, 1094
 - DDS_TopicBuiltinTopicData, 1093
 - DDS_TopicBuiltinTopicDataDataReader, 1093
- Topic Module, 167
- Topic Queries, 703, 911
 - DDS_TOPIC_QUERY_SELECTION_KIND_CONTINUOUS, 708
 - DDS_TOPIC_QUERY_SELECTION_KIND_HISTORY_SNAPSHOT, 708
 - DDS_TOPIC_QUERY_SELECTION_SELECT_ALL, 709
 - DDS_TOPIC_QUERY_SELECTION_USE_READER_CONTENT_FILTER, 709
 - DDS_TopicQuery, 707
 - DDS_TopicQuery_get_guid, 708
 - DDS_TopicQueryData, 707
 - DDS_TopicQueryHelper_topic_query_data_from_service_request, 708
 - DDS_TopicQuerySelection, 706
 - DDS_TopicQuerySelectionKind, 707
- Topic Use Cases, 977
- TOPIC_DATA, 1329
 - DDS_TOPICDATA_QOS_POLICY_NAME, 1330
- topic_data
 - DDS_PublicationBuiltinTopicData, 1910
 - DDS_SubscriptionBuiltinTopicData, 2017
 - DDS_TopicBuiltinTopicData, 2038
 - DDS_TopicQos, 2043
- topic_data_max_length
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1746
- topic_expression
 - DDS_TransportMulticastMapping_t, 2054
- topic_name
 - DDS_PublicationBuiltinTopicData, 1907
 - DDS_SubscriptionBuiltinTopicData, 2014
 - DDS_TopicQueryData, 2046
- TOPIC_QUERY_DISPATCH, 1330
 - DDS_TOPICQUERYDISPATCH_QOS_POLICY_NAME, 1331
- topic_query_dispatch
 - DDS_DataWriterQos, 1673
- topic_query_guid
 - DDS_SampleInfo, 1995
- topic_query_publication_thread
 - DDS_AsynchronousPublisherQosPolicy, 1542
- topic_query_selection
 - DDS_TopicQueryData, 2046
- Topics, 168
 - DDS_ContentFilter_INITIALIZER, 174
 - DDS_ContentFilterCompileFunction, 177

- DDS_ContentFilteredTopic, 177
- DDS_ContentFilteredTopic_append_to_expression_parameters, 203
- DDS_ContentFilteredTopic_as_topicdescription, 201
- DDS_ContentFilteredTopic_get_expression_parameters, 202
- DDS_ContentFilteredTopic_get_filter_expression, 201
- DDS_ContentFilteredTopic_get_related_topic, 206
- DDS_ContentFilteredTopic_narrow, 201
- DDS_ContentFilteredTopic_remove_from_expression_parameters, 204
- DDS_ContentFilteredTopic_set_expression, 203
- DDS_ContentFilteredTopic_set_expression_parameters, 202
- DDS_ContentFilterEvaluateFunction, 178
- DDS_ContentFilterFinalizeFunction, 179
- DDS_ContentFilterWriterAttachFunction, 180
- DDS_ContentFilterWriterCompileFunction, 182
- DDS_ContentFilterWriterDetachFunction, 181
- DDS_ContentFilterWriterEvaluateFunction, 183
- DDS_ContentFilterWriterFinalizeFunction, 180
- DDS_ContentFilterWriterReturnLoanFunction, 183
- DDS_DEFAULT_PRINT_FORMAT, 186
- DDS_FILTERSAMPLEINFO_DEFAULT, 209
- DDS_InconsistentTopicStatus_copy, 187
- DDS_InconsistentTopicStatus_equals, 188
- DDS_InconsistentTopicStatus_finalize, 188
- DDS_InconsistentTopicStatus_initialize, 186
- DDS_InconsistentTopicStatus_INITIALIZER, 173
- DDS_JSON_PRINT_FORMAT, 186
- DDS_MultiTopic, 185
- DDS_MultiTopic_as_topicdescription, 206
- DDS_MultiTopic_get_expression_parameters, 207
- DDS_MultiTopic_get_subscription_expression, 207
- DDS_MultiTopic_narrow, 207
- DDS_MultiTopic_set_expression_parameters, 208
- DDS_PRINT_FORMAT_PROPERTY_DEFAULT, 209
- DDS_PrintFormatKind, 186
- DDS_PrintFormatProperty, 175
- DDS_PrintFormatProperty_INITIALIZER, 172
- DDS_Topic, 175
- DDS_Topic_as_entity, 195
- DDS_Topic_as_topicdescription, 195
- DDS_Topic_get_inconsistent_topic_status, 196
- DDS_Topic_get_listener, 200
- DDS_Topic_get_listenerX, 200
- DDS_Topic_get_qos, 199
- DDS_Topic_narrow, 196
- DDS_Topic_narrow_from_entity, 196
- DDS_Topic_set_listener, 199
- DDS_Topic_set_qos, 197
- DDS_Topic_set_qos_with_profile, 198
- DDS_TopicDescription, 175
- DDS_TopicDescription_get_name, 194
- DDS_TopicDescription_get_participant, 195
- DDS_TopicDescription_get_type_name, 193
- DDS_TopicListener_InconsistentTopicCallback, 176
- DDS_TopicListener_INITIALIZER, 174
- DDS_TopicQos_copy, 193
- DDS_TopicQos_equals, 189
- DDS_TopicQos_finalize, 192
- DDS_TopicQos_initialize, 191
- DDS_TopicQos_INITIALIZER, 173
- DDS_TopicQos_print, 189
- DDS_TopicQos_to_string, 190
- DDS_TopicQos_to_string_w_params, 190
- DDS_XML_PRINT_FORMAT, 186
- total_count
 - DDS_InconsistentTopicStatus, 1809
 - DDS_LivelinessLostStatus, 1822
 - DDS_OfferedDeadlineMissedStatus, 1861
 - DDS_OfferedIncompatibleQosStatus, 1863
 - DDS_PublicationMatchedStatus, 1915
 - DDS_ReliableWriterCacheEventCount, 1946
 - DDS_RequestedDeadlineMissedStatus, 1946
 - DDS_RequestedIncompatibleQosStatus, 1948
 - DDS_SampleLostStatus, 1996
 - DDS_SampleRejectedStatus, 1998
 - DDS_ServiceRequestAcceptedStatus, 2003
 - DDS_SubscriptionMatchedStatus, 2023
- total_count_change
 - DDS_InconsistentTopicStatus, 1809
 - DDS_LivelinessLostStatus, 1823
 - DDS_OfferedDeadlineMissedStatus, 1861
 - DDS_OfferedIncompatibleQosStatus, 1863
 - DDS_PublicationMatchedStatus, 1916
 - DDS_ReliableWriterCacheEventCount, 1946
 - DDS_RequestedDeadlineMissedStatus, 1947
 - DDS_RequestedIncompatibleQosStatus, 1948
 - DDS_SampleLostStatus, 1997
 - DDS_SampleRejectedStatus, 1998
 - DDS_ServiceRequestAcceptedStatus, 2003
 - DDS_SubscriptionMatchedStatus, 2023
- total_samples_dropped_by_instance_replacement
 - DDS_DataReaderCacheStatus, 1590
- trace_file_name
 - DDS_PersistentStorageSettings, 1885
- traffic
 - NDDS_Utility_NetworkCaptureParams_t, 2166
- transfer_mode
 - DDS_DataWriterQos, 1673
- Transport Address, 1027
 - NDDS_Transport_Address_from_string, 1030
 - NDDS_TRANSPORT_ADDRESS_INVALID, 1032
 - NDDS_TRANSPORT_ADDRESS_INVALID_INITIALIZER, 1028
 - NDDS_Transport_Address_is_ipv4, 1031

- NDDS_Transport_Address_is_multicast, 1032
- NDDS_Transport_Address_print, 1031
- NDDS_TRANSPORT_ADDRESS_STRING_BUFFER_SIZE, 1029
- NDDS_Transport_Address_to_string, 1029
- NDDS_Transport_Address_to_string_with_protocol_family_format, 1030
- Transport Plugins Configuration, 1020
- NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT, 1023
- NDDS_TRANSPORT_ALLOCATION_SETTINGS_INCREMENTAL, 1023
- NDDS_TRANSPORT_ALLOCATION_SETTINGS_MAX_TRANSPORT_CLASSID, 1022
- NDDS_TRANSPORT_CLASSID_INVALID, 1023
- NDDS_TRANSPORT_CLASSID_RESERVED_RANGE, 1025
- NDDS_TRANSPORT_CLASSID_SHMEM, 1024
- NDDS_TRANSPORT_CLASSID_SHMEM_510, 1024
- NDDS_Transport_ClassId_t, 1026
- NDDS_TRANSPORT_CLASSID_TCPV4_LAN, 1024
- NDDS_TRANSPORT_CLASSID_TCPV4_WAN, 1024
- NDDS_TRANSPORT_CLASSID_TLSV4_LAN, 1025
- NDDS_TRANSPORT_CLASSID_TLSV4_WAN, 1025
- NDDS_TRANSPORT_CLASSID_UDPv4, 1023
- NDDS_TRANSPORT_CLASSID_UDPv4_WAN, 1025
- NDDS_TRANSPORT_CLASSID_UDPv6, 1024
- NDDS_TRANSPORT_CLASSID_UDPv6_510, 1024
- NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN, 1025
- NDDS_TRANSPORT_INTERFACE_RANK_UNKNOWN, 1022
- NDDS_TRANSPORT_LENGTH_UNLIMITED, 1022
- NDDS_TRANSPORT_PORT_INVALID, 1022
- NDDS_Transport_Port_t, 1026
- NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS, 1025
- NDDS_TRANSPORT_PROPERTY_GATHER_SEND_BUFFER_SIZE, 1026
- NDDS_TRANSPORT_UUID_SIZE, 1022
- NDDS_TRANSPORT_UUID_UNKNOWN, 1022
- Transport Use Cases, 992
- TRANSPORT_BUILTIN, 1331
 - DDS_TRANSPORTBUILTIN_MASK_ALL, 1333
 - DDS_TRANSPORTBUILTIN_MASK_DEFAULT, 1332
 - DDS_TRANSPORTBUILTIN_MASK_NONE, 1332
 - DDS_TRANSPORTBUILTIN_QOS_POLICY_NAME, 1334
 - DDS_TRANSPORTBUILTIN_SHMEM, 1334
 - DDS_TRANSPORTBUILTIN_SHMEM_ALIAS, 1334
 - DDS_TRANSPORTBUILTIN_UDPv4, 1334
 - DDS_TRANSPORTBUILTIN_UDPv4_ALIAS, 1334
 - DDS_TRANSPORTBUILTIN_UDPv4_WAN, 1334
 - DDS_TRANSPORTBUILTIN_UDPv4_WAN_ALIAS, 1334
 - DDS_TRANSPORTBUILTIN_UDPv6, 1334
 - DDS_TRANSPORTBUILTIN_UDPv6_ALIAS, 1334
 - DDS_TransportBuiltinKind, 1333
 - DDS_TransportBuiltinKindMask, 1333
 - TRANSPORTBUILTIN_COUNT_AUTOMATIC, 1334
 - DDS_DomainParticipantQos, 1729
 - TRANSPORT_CLASSID_UNLIMITED, 1334
 - NDDS_Transport_Interface_t, 2121
 - transport_common.h, 2184
 - transport_common_user.h, 2189
 - transport_info
 - DDS_ParticipantBuiltinTopicData, 1873
 - transport_info_list_max_length
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1755
 - transport_interface_user.h, 2198
 - TRANSPORT_MULTICAST, 1335
 - DDS_AUTOMATIC_TRANSPORT_MULTICAST_QOS, 1336
 - DDS_TRANSPORTMULTICAST_QOS_POLICY_NAME, 1336
 - DDS_TransportMulticastQosPolicyKind, 1336
 - DDS_UNICAST_ONLY_TRANSPORT_MULTICAST_QOS, 1336
 - TRANSPORT_MULTICAST_MAPPING, 1336
 - DDS_TRANSPORTMULTICASTMAPPING_QOS_POLICY_NAME, 1337
 - TRANSPORT_PRIORITY, 1337
 - DDS_TRANSPORTPRIORITY_QOS_POLICY_NAME, 1338
 - transport_priority
 - DDS_DataReaderQos, 1617
 - DDS_DataWriterQos, 1669
 - DDS_TransportBuiltinTopicData, 2037
 - DDS_TopicQos, 2045
 - TRANSPORT_PRIORITY_MAPPING_high
 - NDDS_Transport_UDPv4_Property_t, 2140
 - NDDS_Transport_UDPv4_WAN_Property_t, 2149
 - NDDS_Transport_UDPv6_Property_t, 2160
 - transport_priority_mapping_low
 - NDDS_Transport_UDPv4_Property_t, 2140
 - NDDS_Transport_UDPv4_WAN_Property_t, 2149
 - NDDS_Transport_UDPv6_Property_t, 2160
 - transport_priority_mask
 - NDDS_Transport_UDPv4_Property_t, 2140
 - NDDS_Transport_UDPv4_WAN_Property_t, 2148
 - NDDS_Transport_UDPv6_Property_t, 2159
 - TRANSPORT_SELECTION, 1338

- DDS_TRANSPORTSELECTION_QOS_POLICY_NAME, 1338
- transport_selection
 - DDS_DataReaderQos, 1616
 - DDS_DataWriterQos, 1671
- transport_shmem.h, 2201
- transport_udp.h, 2205
- transport_udpv4.h, 2217
- transport_udpv6.h, 2229
- TRANSPORT_UNICAST, 1339
 - DDS_TRANSPORTUNICAST_QOS_POLICY_NAME, 1339
- transport_uuid
 - NDDS_Transport_Property_t, 2128
- TransportAllocationSettings_t, 2181
- Transports, 720, 908
- transports
 - DDS_TransportMulticastSettings_t, 2061
 - DDS_TransportUnicastSettings_t, 2068
 - NDDS_Utility_NetworkCaptureParams_t, 2165
- trim_to_size
 - DDS_DynamicDataTypeSerializationProperty_t, 1783
- trust_algorithm_info
 - DDS_ParticipantBuiltinTopicData, 1874
 - DDS_PublicationBuiltinTopicData, 1913
 - DDS_SubscriptionBuiltinTopicData, 2020
- trust_chain
 - DDS_ParticipantTrustSignatureAlgorithmInfo, 1881
- trust_protection_info
 - DDS_ParticipantBuiltinTopicData, 1874
 - DDS_PublicationBuiltinTopicData, 1913
 - DDS_SubscriptionBuiltinTopicData, 2020
- type
 - DDS_StructMember, 2007
 - DDS_UnionMember, 2085
 - DDS_ValueMember, 2089
- Type Code Support, 228
 - DDS_ExtensibilityKind, 243
 - DDS_EXTENSIBLE_EXTENSIBILITY, 244
 - DDS_FINAL_EXTENSIBILITY, 244
 - DDS_g_tc_boolean, 302
 - DDS_g_tc_char, 303
 - DDS_g_tc_double, 302
 - DDS_g_tc_float, 302
 - DDS_g_tc_int8, 303
 - DDS_g_tc_long, 301
 - DDS_g_tc_longdouble, 305
 - DDS_g_tc_longlong, 304
 - DDS_g_tc_null, 300
 - DDS_g_tc_octet, 303
 - DDS_g_tc_short, 300
 - DDS_g_tc_uint8, 304
 - DDS_g_tc_ulong, 301
 - DDS_g_tc_ulonglong, 304
 - DDS_g_tc_ushort, 301
 - DDS_g_tc_wchar, 305
 - DDS_MUTABLE_EXTENSIBILITY, 244
 - DDS_PRIVATE_MEMBER, 239
 - DDS_PUBLIC_MEMBER, 240
 - DDS_TCKind, 243
 - DDS_TK_ALIAS, 243
 - DDS_TK_ARRAY, 243
 - DDS_TK_BOOLEAN, 243
 - DDS_TK_CHAR, 243
 - DDS_TK_DOUBLE, 243
 - DDS_TK_ENUM, 243
 - DDS_TK_FLOAT, 243
 - DDS_TK_INT8, 243
 - DDS_TK_LONG, 243
 - DDS_TK_LONGDOUBLE, 243
 - DDS_TK_LONGLONG, 243
 - DDS_TK_NULL, 243
 - DDS_TK_OCTET, 243
 - DDS_TK_SEQUENCE, 243
 - DDS_TK_SHORT, 243
 - DDS_TK_STRING, 243
 - DDS_TK_STRUCT, 243
 - DDS_TK_UINT8, 243
 - DDS_TK_ULONG, 243
 - DDS_TK_ULONGLONG, 243
 - DDS_TK_UNION, 243
 - DDS_TK_USHORT, 243
 - DDS_TK_VALUE, 243
 - DDS_TK_WCHAR, 243
 - DDS_TK_WSTRING, 243
 - DDS_TYPE_CODE_PRINT_KIND_IDL, 243
 - DDS_TYPE_CODE_PRINT_KIND_XML, 243
 - DDS_TypeCode_add_member, 280
 - DDS_TypeCode_add_member_ex, 282
 - DDS_TypeCode_add_member_to_enum, 278
 - DDS_TypeCode_add_member_to_union, 279
 - DDS_TypeCode_array_dimension, 268
 - DDS_TypeCode_array_dimension_count, 268
 - DDS_TypeCode_cdr_serialized_sample_key_max_size, 288
 - DDS_TypeCode_cdr_serialized_sample_max_size, 287
 - DDS_TypeCode_cdr_serialized_sample_min_size, 288
 - DDS_TypeCode_concrete_base_type, 272
 - DDS_TypeCode_content_type, 270
 - DDS_TypeCode_default_annotation, 249
 - DDS_TypeCode_default_index, 271
 - DDS_TypeCode_default_value, 250
 - DDS_TypeCode_discriminator_type, 266
 - DDS_TypeCode_element_count, 269
 - DDS_TypeCode_equal, 247

- DDS_TypeCode_extensibility_kind, 245
- DDS_TypeCode_find_member_by_id, 275
- DDS_TypeCode_find_member_by_name, 256
- DDS_TypeCode_get_cdr_serialized_sample_max_size, 286
- DDS_TypeCode_get_type_object_serialized_size, 277
- DDS_TYPECODE_INDEX_INVALID, 238
- DDS_TypeCode_is_alias_pointer, 270
- DDS_TypeCode_is_member_bitfield, 263
- DDS_TypeCode_is_member_key, 261
- DDS_TypeCode_is_member_pointer, 262
- DDS_TypeCode_is_member_required, 262
- DDS_TYPECODE_KEY_MEMBER, 240
- DDS_TypeCode_kind, 244
- DDS_TypeCode_length, 267
- DDS_TypeCode_max_annotation, 250
- DDS_TypeCode_max_value, 252
- DDS_TypeCode_member_bitfield_bits, 264
- DDS_TypeCode_member_count, 255
- DDS_TypeCode_member_default_annotation, 249
- DDS_TypeCode_member_default_value, 252
- DDS_TypeCode_member_id, 275
- DDS_TYPECODE_MEMBER_ID_INVALID, 237
- DDS_TypeCode_member_label, 259
- DDS_TypeCode_member_label_count, 258
- DDS_TypeCode_member_max_value, 254
- DDS_TypeCode_member_min_value, 253
- DDS_TypeCode_member_name, 256
- DDS_TypeCode_member_ordinal, 260
- DDS_TypeCode_member_type, 257
- DDS_TypeCode_member_visibility, 265
- DDS_TypeCode_min_annotation, 250
- DDS_TypeCode_min_value, 251
- DDS_TypeCode_name, 248
- DDS_TYPECODE_NONKEY_MEMBER, 240
- DDS_TYPECODE_NONKEY_REQUIRED_MEMBER, 241
- DDS_TYPECODE_NOT_BITFIELD, 238
- DDS_TypeCode_print, 284
- DDS_TypeCode_print_IDL, 283
- DDS_TypeCode_PrintFormat_INITIALIZER, 241
- DDS_TypeCode_to_string, 285
- DDS_TypeCode_to_string_w_format, 286
- DDS_TypeCode_type_modifier, 274
- DDS_TypeCodeFactory_clone_tc, 289
- DDS_TypeCodeFactory_create_alias_tc, 297
- DDS_TypeCodeFactory_create_array_tc, 300
- DDS_TypeCodeFactory_create_enum_tc, 296
- DDS_TypeCodeFactory_create_enum_tc_ex, 296
- DDS_TypeCodeFactory_create_sequence_tc, 299
- DDS_TypeCodeFactory_create_string_tc, 298
- DDS_TypeCodeFactory_create_struct_tc, 291
- DDS_TypeCodeFactory_create_struct_tc_ex, 292
- DDS_TypeCodeFactory_create_union_tc, 294
- DDS_TypeCodeFactory_create_union_tc_ex, 295
- DDS_TypeCodeFactory_create_value_tc, 293
- DDS_TypeCodeFactory_create_value_tc_ex, 293
- DDS_TypeCodeFactory_create_wstring_tc, 298
- DDS_TypeCodeFactory_delete_tc, 290
- DDS_TypeCodeFactory_finalize_instance, 289
- DDS_TypeCodeFactory_get_instance, 289
- DDS_TypeCodeFactory_get_primitive_tc, 291
- DDS_TypeCodePrintFormatKind, 242
- DDS_ValueModifier, 242
- DDS_Visibility, 242
- DDS_VM_ABSTRACT, 239
- DDS_VM_CUSTOM, 238
- DDS_VM_NONE, 238
- DDS_VM_TRUNCATABLE, 239
- Type System, 912
- type_code
 - DDS_PublicationBuiltinTopicData, 1910
 - DDS_SubscriptionBuiltinTopicData, 2018
- type_code_max_serialized_length
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1749
- type_consistency
 - DDS_DataReaderQos, 1615
 - DDS_SubscriptionBuiltinTopicData, 2017
- TYPE_CONSISTENCY_ENFORCEMENT, 1339
 - DDS_ALLOW_TYPE_COERCION, 1341
 - DDS_AUTO_TYPE_COERCION, 1341
 - DDS_DISALLOW_TYPE_COERCION, 1341
 - DDS_TYPE_CONSISTENCY_ENFORCEMENT_QOS_POLICY_NAME, 1341
 - DDS_TypeConsistencyKind, 1340
- type_name
 - DDS_PublicationBuiltinTopicData, 1907
 - DDS_SubscriptionBuiltinTopicData, 2014
 - DDS_TopicBuiltinTopicData, 2036
- type_object_max_deserialized_length
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1750
- type_object_max_serialized_length
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1749
- type_support
 - DDS_DataReaderQos, 1617
 - DDS_DataWriterQos, 1673
 - DDS_DomainParticipantQos, 1731
- TYPESUPPORT, 1341
 - DDS_AUTO_CDR_PADDING, 1342
 - DDS_CdrPaddingKind, 1342
 - DDS_NOT_SET_CDR_PADDING, 1342
 - DDS_TYPESUPPORT_QOS_POLICY_NAME, 1343
 - DDS_ZERO_CDR_PADDING, 1342

UDP Transport Plugin definitions, 1033
 NDDS_TRANSPORT_UDP_GATHER_SEND_BUFFER_COUNT_DEFAULT, 1048
 1034
 NDDS_TRANSPORT_UDP_MULTICAST_TTL_DEFAULT, 1046
 1035
 NDDS_Transport_UDP_Port, 1035
 NDDS_TRANSPORT_UDP_PROPERTIES_BITMAP_DEFAULT, 1062
 1033
 NDDS_TRANSPORT_UDP_RECV_SOCKET_BUFFER_SIZE_DEFAULT, 1064
 1035
 NDDS_TRANSPORT_UDP_SEND_SOCKET_BUFFER_SIZE_DEFAULT, 1064
 1034
 NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_DEFAULT, 1067
 1034
 NDDS_TRANSPORT_UDP_SOCKET_BUFFER_SIZE_OS_MAX, 1062
 1034
 udp_debugging_address
 NDDS_Transport_Shmem_Property_t, 2131
 udp_debugging_port
 NDDS_Transport_Shmem_Property_t, 2132
 UDPv4 Transport, 1041
 NDDS_TRANSPORT_UDPv4_ADDRESS_BIT_COUNT, 1063
 1046
 NDDS_TRANSPORT_UDPv4_BLOCKING_ALWAYS, 1062
 1048
 NDDS_TRANSPORT_UDPv4_BLOCKING_DEFAULT, 1063
 1048
 NDDS_TRANSPORT_UDPv4_BLOCKING_NEVER, 1063
 1048
 NDDS_Transport_UDPv4_create, 1050
 NDDS_Transport_UDPv4_create_from_properties_with_prefix, 1063
 1051
 NDDS_TRANSPORT_UDPv4_GATHER_SEND_BUFFER_COUNT_DEFAULT, 1062
 1046
 NDDS_TRANSPORT_UDPv4_MESSAGE_SIZE_MAX_DEFAULT, 1064
 1047
 NDDS_TRANSPORT_UDPv4_MULTICAST_TTL_DEFAULT, 1064
 1048
 NDDS_Transport_UDPv4_new, 1049
 NDDS_TRANSPORT_UDPv4_PAYLOAD_SIZE_MAX, 1069
 1047
 NDDS_TRANSPORT_UDPv4_PROPERTIES_BITMAP_DEFAULT, 1067
 1046
 NDDS_TRANSPORT_UDPv4_PROPERTY_DEFAULT, 1067
 1048
 NDDS_TRANSPORT_UDPv4_RECV_SOCKET_BUFFER_SIZE_DEFAULT, 1064
 1047
 NDDS_TRANSPORT_UDPv4_SEND_SOCKET_BUFFER_SIZE_DEFAULT, 1064
 1047
 NDDS_TRANSPORT_UDPv4_SOCKET_BUFFER_SIZE_OS_DEFAULT, 1067
 1047
 NDDS_TRANSPORT_UDPv4_SOCKET_BUFFER_SIZE_OS_MAX, 1067
 1046
 NDDS_Transport_UDPv4_string_to_address_cEA, 1064
 1048

- url_profile
 - DDS_ProfileQosPolicy, 1899
- use_42e_compatible_alignment
 - DDS_DynamicDataTypeSerializationProperty_t, 1782
- use_checksum
 - NDDS_Transport_UDPv4_Property_t, 2139
 - NDDS_Transport_UDPv4_WAN_Property_t, 2148
- User Data Type Support, 210
 - DDS_DATAREADER_C, 213
 - DDS_DATAWRITER_C, 213
 - DDS_HANDLE_NIL, 228
 - DDS_InstanceHandle_compare, 227
 - DDS_InstanceHandle_copy, 227
 - DDS_InstanceHandle_equals, 226
 - DDS_InstanceHandle_is_nil, 228
 - DDS_InstanceHandle_t, 214
 - DDS_TypeSupport, 214
 - DDS_TYPESUPPORT_C, 213
 - FooTypeSupport_copy_data, 216
 - FooTypeSupport_create_data, 215
 - FooTypeSupport_create_data_ex, 215
 - FooTypeSupport_create_data_w_params, 216
 - FooTypeSupport_data_to_string, 225
 - FooTypeSupport_delete_data, 217
 - FooTypeSupport_delete_data_ex, 217
 - FooTypeSupport_delete_data_w_params, 218
 - FooTypeSupport_deserialize_data_from_cdr_buffer, 225
 - FooTypeSupport_finalize_data, 220
 - FooTypeSupport_finalize_data_ex, 220
 - FooTypeSupport_get_type_name, 221
 - FooTypeSupport_get_typecode, 226
 - FooTypeSupport_initialize_data, 218
 - FooTypeSupport_initialize_data_ex, 219
 - FooTypeSupport_print_data, 223
 - FooTypeSupport_register_type, 221
 - FooTypeSupport_serialize_data_to_cdr_buffer, 223
 - FooTypeSupport_serialize_data_to_cdr_buffer_ex, 224
 - FooTypeSupport_unregister_type, 222
- User-managed Threads, 1432
 - DDS_ThreadFactory_CreateThreadCallback, 1433
 - DDS_ThreadFactory_DeleteThreadCallback, 1434
 - DDS_ThreadFactory_INITIALIZER, 1433
 - DDS_ThreadFactory_OnSpawnedFunction, 1433
- USER_DATA, 1343
 - DDS_USERDATA_QOS_POLICY_NAME, 1343
- user_data
 - DDS_DataReaderQos, 1614
 - DDS_DataWriterQos, 1670
 - DDS_DomainParticipantQos, 1728
 - DDS_ParticipantBuiltinTopicData, 1871
 - DDS_PublicationBuiltinTopicData, 1909
 - DDS_SubscriptionBuiltinTopicData, 2016
 - RTI_Connext_ReplierListener, 2168
 - RTI_Connext_SimpleReplierListener, 2177
- user_endpoints_default_required_mask
 - DDS_ParticipantTrustInterceptorAlgorithmInfo, 1878
- user_forwarding_level
 - DDS_MonitoringLoggingForwardingSettings, 1848
- user_multicast_port_offset
 - DDS_RtpsWellKnownPorts_t, 1983
- user_unicast_port_offset
 - DDS_RtpsWellKnownPorts_t, 1983
- Utilities, 776
 - RTI_Connext_Messaging_get_api_build_number_string, 777
 - RTI_Connext_Messaging_get_api_version, 776
 - RTI_Connext_Messaging_Library_get_api_version_string, 777
 - RTI_Connext_Messaging_LibraryVersion, 776
- vacuum
 - DDS_PersistentStorageSettings, 1886
- valid_data
 - DDS_SampleInfo, 1992
- valid_response_data
 - DDS_AcknowledgmentInfo, 1536
- value
 - Common types and functions, 1111
 - DDS_AckResponseData_t, 1537
 - DDS_Cookie_t, 1580
 - DDS_DataRepresentationQosPolicy, 1640
 - DDS_GroupDataQosPolicy, 1804
 - DDS_GUID_t, 1805
 - DDS_OwnershipStrengthQosPolicy, 1869
 - DDS_Property_t, 1901
 - DDS_PropertyQosPolicy, 1903
 - DDS_Tag, 2027
 - DDS_TopicDataQosPolicy, 2041
 - DDS_TransportMulticastMappingQosPolicy, 2058
 - DDS_TransportMulticastQosPolicy, 2059
 - DDS_TransportPriorityQosPolicy, 2064
 - DDS_TransportUnicastQosPolicy, 2067
 - DDS_UserDataQosPolicy, 2087
 - KeyedOctets Built-in Type, 1199
 - KeyedString Built-in Type, 1151
 - Octets Built-in Type, 1169
- vendorId
 - DDS_VendorId_t, 2091
- verbosity
 - DDS_LoggingQosPolicy, 1832
- Version, 1436
 - NDDS_Config_Version_get_api_version, 1437
 - NDDS_Config_Version_get_core_version, 1438
 - NDDS_Config_Version_get_product_version, 1437
 - NDDS_Config_Version_to_string, 1438

- View States, 711
 - DDS_ANY_VIEW_STATE, 713
 - DDS_NEW_VIEW_STATE, 713
 - DDS_NOT_NEW_VIEW_STATE, 713
 - DDS_ViewStateKind, 712
 - DDS_ViewStateMask, 712
- view_state
 - DDS_SampleInfo, 1989
- view_states
 - DDS_ReadConditionParams, 1930
- virtual_duplicate_dropped_sample_count
 - DDS_DataReaderCacheStatus, 1589
- virtual_guid
 - DDS_DataReaderProtocolQosPolicy, 1597
 - DDS_DataWriterProtocolQosPolicy, 1650
 - DDS_PublicationBuiltinTopicData, 1911
 - DDS_SubscriptionBuiltinTopicData, 2019
- virtual_heartbeat_period
 - DDS_RtpsReliableWriterProtocol_t, 1965
- wait_timeout
 - DDS_AsyncWaitSetProperty_t, 1546
- Waitset Use Cases, 991
- waitset_property
 - DDS_AsyncWaitSetProperty_t, 1545
- WIRE_PROTOCOL, 1344
 - DDS_INTEROPERABLE_RTPS_WELL_KNOWN_PORTS, 1349
 - DDS_RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS, 1348
 - DDS_RTPS_AUTO_ID_FROM_IP, 1348
 - DDS_RTPS_AUTO_ID_FROM_MAC, 1348
 - DDS_RTPS_AUTO_ID_FROM_UUID, 1348
 - DDS_RTPS_RESERVED_PORT_BUILTIN_MULTICAST, 1348
 - DDS_RTPS_RESERVED_PORT_BUILTIN_UNICAST, 1348
 - DDS_RTPS_RESERVED_PORT_MASK_ALL, 1346
 - DDS_RTPS_RESERVED_PORT_MASK_DEFAULT, 1345
 - DDS_RTPS_RESERVED_PORT_MASK_NONE, 1346
 - DDS_RTPS_RESERVED_PORT_USER_MULTICAST, 1348
 - DDS_RTPS_RESERVED_PORT_USER_UNICAST, 1348
 - DDS_RtpsReservedPortKind, 1347
 - DDS_RtpsReservedPortKindMask, 1347
 - DDS_WIREPROTOCOL_QOS_POLICY_NAME, 1349
 - DDS_WireProtocolQosPolicyAutoKind, 1348
- wire_protocol
 - DDS_DomainParticipantQos, 1728
- write
 - NDDS_Config_LoggerDevice, 2117
 - WriteParams, 1389
 - DDS_AUTO_SAMPLE_IDENTITY, 1391
 - DDS_SampleIdentity_equals, 1389
 - DDS_UNKNOWN_SAMPLE_IDENTITY, 1392
 - DDS_WRITEPARAMS_DEFAULT, 1392
 - DDS_WriteParams_reset, 1391
 - sequence_number, 1391
 - writer_guid, 1391
 - writer_attach
 - DDS_ContentFilter, 1577
 - writer_compile
 - DDS_ContentFilter, 1573
 - writer_compression_level
 - DDS_CompressionSettings_t, 1569
 - writer_compression_threshold
 - DDS_CompressionSettings_t, 1569
 - WRITER_DATA_LIFECYCLE, 1344
 - DDS_WRITERDATALIFECYCLE_QOS_POLICY_NAME, 1344
 - writer_data_lifecycle
 - DDS_DataWriterQos, 1670
 - writer_data_tag_list_max_length
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1757
 - writer_data_tag_string_max_length
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1757
 - writer_depth
 - DDS_DurabilityQosPolicy, 1762
 - writer_detach
 - DDS_ContentFilter, 1578
 - writer_evaluate
 - DDS_ContentFilter, 1576
 - writer_finalize
 - DDS_ContentFilter, 1577
 - writer_guid
 - WriteParams, 1391
 - writer_instance_cache_allocation
 - DDS_PersistentStorageSettings, 1887
 - writer_loaned_sample_allocation
 - DDS_DataWriterResourceLimitsQosPolicy, 1681
 - writer_memory_state
 - DDS_PersistentStorageSettings, 1889
 - writer_property_list_max_length
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1753
 - writer_property_string_max_length
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1753
 - writer_removed_batch_sample_dropped_sample_count
 - DDS_DataReaderCacheStatus, 1590
 - writer_resource_limits
 - DDS_DataWriterQos, 1671

- writer_return_loan
 - DDS_ContentFilter, 1578
- writer_sample_cache_allocation
 - DDS_PersistentStorageSettings, 1888
- writer_side_filter_optimization
 - DDS_ExpressionProperty, 1795
- writer_user_data_max_length
 - DDS_DomainParticipantResourceLimitsQosPolicy, 1747

Zero Copy Transfer Over Shared Memory, 210