RTI Data Distribution Service .Net APIs

Version 4.5c

Generated by Doxygen 1.5.5

Wed Jun 9 20:15:25 2010

Contents

1 RTI Data Distribution Service **1**

1.1 Feedback and Support for this Release. 1

1.2 Available Documentation. 2

2 Module Index **5**

2.1 Modules 5

3 Class Index **9**

3.1 Class Hierarchy 9

4 Class Index **17**

4.1 Class List 17

5 Module Documentation **31**

5.1 Clock Selection 31

5.2 Domain Module 34

5.3 DomainParticipantFactory 35

5.4 DomainParticipants 37

5.5 Built-in Topics 42

5.6 Topic Module 50

5.7 Topics 51

5.8 User Data Type Support 52

5.9 Type Code Support 55

5.10 Built-in Types 68

5.11 Dynamic Data . . . 73
5.12 Publication Module . . . 78
5.13 Publishers . . . 79
5.14 Data Writers . . . 82
5.15 Flow Controllers . . . 84
5.16 Subscription Module . . . 91
5.17 Subscribers . . . 94
5.18 DataReaders . . . 97
5.19 Read Conditions . . . 100
5.20 Query Conditions . . . 101
5.21 Data Samples . . . 102
5.22 Sample States . . . 103
5.23 View States . . . 104
5.24 Instance States . . . 105
5.25 Infrastructure Module . . . 107
5.26 Built-in Sequences . . . 109
5.27 Multi-channel DataWriters . . . 111
5.28 Pluggable Transports . . . 113
5.29 Using Transport Plugins . . . 119
5.30 Built-in Transport Plugins . . . 122
5.31 Configuration Utilities . . . 124
5.32 Unsupported Utilities . . . 128
5.33 Durability and Persistence . . . 129
5.34 Configuring QoS Profiles with XML . . . 135
5.35 Publication Example . . . 138
5.36 Subscription Example . . . 139
5.37 Participant Use Cases . . . 140
5.38 Topic Use Cases . . . 143
5.39 FlowController Use Cases . . . 145
5.40 Publisher Use Cases . . . 149
5.41 DataWriter Use Cases . . . 150

5.42 Subscriber Use Cases . . . 152
5.43 DataReader Use Cases . . . 155
5.44 Entity Use Cases . . . 160
5.45 Waitset Use Cases . . . 163
5.46 Transport Use Cases . . . 165
5.47 Filter Use Cases . . . 166
5.48 Large Data Use Cases . . . 171
5.49 Documentation Roadmap . . . 173
5.50 Conventions . . . 174
5.51 .Net Language Support . . . 177
5.52 DDS API Reference . . . 179
5.53 Queries and Filters Syntax . . . 184
5.54 RTI Data Distribution Service API Reference . . . 192
5.55 Programming How-To's . . . 193
5.56 Programming Tools . . . 195
5.57 rtiddsgen . . . 196
5.58 rtiddsping . . . 208
5.59 rtiddsspy . . . 215
5.60 Octets Built-in Type . . . 222
5.61 KeyedOctets Built-in Type . . . 223
5.62 KeyedString Built-in Type . . . 224
5.63 String Built-in Type . . . 225
5.64 Participant Built-in Topics . . . 226
5.65 Topic Built-in Topics . . . 228
5.66 Publication Built-in Topics . . . 230
5.67 Subscription Built-in Topics . . . 232
5.68 Return Codes . . . 234
5.69 Status Kinds . . . 238
5.70 Exception Codes . . . 248
5.71 Time Support . . . 250
5.72 GUID Support . . . 254

5.73 Sequence Number Support 255
5.74 Thread Settings 257
5.75 QoS Policies 260
5.76 USER_DATA 273
5.77 TOPIC_DATA 274
5.78 GROUP_DATA 275
5.79 DURABILITY 276
5.80 PRESENTATION 279
5.81 DEADLINE 281
5.82 LATENCY_BUDGET 282
5.83 OWNERSHIP 283
5.84 OWNERSHIP_STRENGTH 285
5.85 LIVELINESS 286
5.86 TIME_BASED_FILTER 288
5.87 PARTITION 289
5.88 RELIABILITY 290
5.89 DESTINATION_ORDER 292
5.90 HISTORY 294
5.91 DURABILITY_SERVICE 297
5.92 RESOURCE_LIMITS 298
5.93 TRANSPORT_PRIORITY 300
5.94 LIFESPAN 301
5.95 WRITER_DATA_LIFECYCLE 302
5.96 READER_DATA_LIFECYCLE 303
5.97 ENTITY_FACTORY 304
5.98 Extended Qos Support 305
5.99 Unicast Settings 306
5.100Multicast Settings 307
5.101TRANSPORT_SELECTION 308
5.102TRANSPORT_UNICAST 309
5.103TRANSPORT_MULTICAST 310

5.104NDDS_DISCOVERY_PEERS 312
5.105DISCOVERY 320
5.106TRANSPORT_BUILTIN 321
5.107WIRE_PROTOCOL 325
5.108DATA_READER_RESOURCE_LIMITS 331
5.109DATA_WRITER_RESOURCE_LIMITS 333
5.110DATA_READER_PROTOCOL 337
5.111DATA_WRITER_PROTOCOL 338
5.112SYSTEM_RESOURCE_LIMITS 339
5.113DOMAIN_PARTICIPANT_RESOURCE_LIMITS 340
5.114EVENT 341
5.115DATABASE 342
5.116RECEIVER_POOL 343
5.117PUBLISH_MODE 344
5.118DISCOVERY_CONFIG 347
5.119TYPESUPPORT 350
5.120ASYNCHRONOUS_PUBLISHER 351
5.121EXCLUSIVE_AREA 352
5.122BATCH 353
5.123LOCATORFILTER 354
5.124MULTICHANNEL 356
5.125PROPERTY 357
5.126Entity Support 363
5.127ENTITY_NAME 364
5.128PROFILE 365
5.129Conditions and WaitSets 366
5.130Sequence Support 367

6 Class Documentation **369**

6.1 DDS::AllocationSettings_t Struct Reference 369
6.2 DDS::AsynchronousPublisherQosPolicy Class Reference 371

6.3 DDS::BatchQosPolicy Struct Reference . . . 376
6.4 DDS::BooleanSeq Class Reference . . . 381
6.5 DDS::BuiltinTopicKey_t Struct Reference . . . 383
6.6 DDS::BuiltinTopicReaderResourceLimits_t Struct Reference . . . 385
6.7 DDS::Bytes Struct Reference . . . 388
6.8 DDS::BytesDataReader Class Reference . . . 391
6.9 DDS::BytesDataWriter Class Reference . . . 392
6.10 DDS::ByteSeq Class Reference . . . 395
6.11 DDS::BytesSeq Class Reference . . . 397
6.12 DDS::BytesTypeSupport Class Reference . . . 399
6.13 DDS::ChannelSettings_t Class Reference . . . 403
6.14 DDS::ChannelSettingsSeq Class Reference . . . 405
6.15 DDS::CharSeq Class Reference . . . 406
6.16 DDS::Condition Class Reference . . . 408
6.17 DDS::ConditionSeq Class Reference . . . 410
6.18 NDDS::Config_LibraryVersion_t Struct Reference . . . 411
6.19 NDDS::ConfigLogger Class Reference . . . 413
6.20 NDDS::ConfigVersion Class Reference . . . 417
6.21 DDS::ContentFilteredTopic Class Reference . . . 419
6.22 DDS::ContentFilterProperty_t Class Reference . . . 426
6.23 DDS::DatabaseQosPolicy Class Reference . . . 428
6.24 DDS::DataReader Class Reference . . . 433
6.25 DDS::DataReaderCacheStatus Struct Reference . . . 460
6.26 DDS::DataReaderListener Class Reference . . . 461
6.27 DDS::DataReaderProtocolQosPolicy Struct Reference . . . 465
6.28 DDS::DataReaderProtocolStatus Struct Reference . . . 470
6.29 DDS::DataReaderQos Class Reference . . . 480
6.30 DDS::DataReaderResourceLimitsQosPolicy Struct Reference . . . 486
6.31 DDS::DataReaderSeq Class Reference . . . 498
6.32 DDS::DataWriter Class Reference . . . 499
6.33 DDS::DataWriterCacheStatus Struct Reference . . . 523

6.34 DDS::DataWriterListener Class Reference 524
6.35 DDS::DataWriterProtocolQosPolicy Struct Reference 529
6.36 DDS::DataWriterProtocolStatus Struct Reference 534
6.37 DDS::DataWriterQos Class Reference 546
6.38 DDS::DataWriterResourceLimitsQosPolicy Struct Reference . . . 552
6.39 DDS::DeadlineQosPolicy Struct Reference 557
6.40 DDS::DestinationOrderQosPolicy Struct Reference 560
6.41 DDS::DiscoveryConfigQosPolicy Class Reference 563
6.42 DDS::DiscoveryQosPolicy Class Reference 571
6.43 DDS::DomainEntity Class Reference 576
6.44 DDS::DomainParticipant Class Reference 577
6.45 DDS::DomainParticipantFactory Class Reference 649
6.46 DDS::DomainParticipantFactoryQos Class Reference 673
6.47 DDS::DomainParticipantListener Class Reference 675
6.48 DDS::DomainParticipantQos Class Reference 683
6.49 DDS::DomainParticipantResourceLimitsQosPolicy Struct Reference 688
6.50 DDS::DoubleSeq Class Reference 707
6.51 DDS::DurabilityQosPolicy Struct Reference 709
6.52 DDS::DurabilityServiceQosPolicy Struct Reference 714
6.53 DDS::Duration_t Struct Reference 717
6.54 DDS::DynamicData Class Reference 719
6.55 DDS::DynamicDataInfo Class Reference 809
6.56 DDS::DynamicDataMemberInfo Class Reference 810
6.57 DDS::DynamicDataProperty_t Class Reference 813
6.58 DDS::DynamicDataReader Class Reference 815
6.59 DDS::DynamicDataSeq Class Reference 816
6.60 DDS::DynamicDataTypeProperty_t Class Reference 818
6.61 DDS::DynamicDataTypeSerializationProperty_t Class Reference 820
6.62 DDS::DynamicDataTypeSupport Class Reference 822
6.63 DDS::DynamicDataWriter Class Reference 828

6.64 DDS::Entity Class Reference 845
6.65 DDS::EntityFactoryQosPolicy Struct Reference 851
6.66 DDS::EntityNameQosPolicy Class Reference 854
6.67 DDS::EnumMember Class Reference 856
6.68 DDS::EnumMemberSeq Class Reference 857
6.69 DDS::EventQosPolicy Class Reference 858
6.70 DDS::Exception Class Reference 861
6.71 DDS::ExclusiveAreaQosPolicy Struct Reference 862
6.72 DDS::FloatSeq Class Reference 865
6.73 DDS::FlowController Class Reference 867
6.74 DDS::FlowControllerProperty_t Class Reference 871
6.75 DDS::FlowControllerTokenBucketProperty_t Struct Reference . . 873
6.76 Foo Struct Reference 877
6.77 FooDataReader Class Reference 878
6.78 FooDataWriter Class Reference 879
6.79 FooSeq Class Reference 880
6.80 FooTypeSupport Class Reference 884
6.81 DDS::GroupDataQosPolicy Class Reference 890
6.82 DDS::GuardCondition Class Reference 892
6.83 DDS::GUID_t Struct Reference 894
6.84 DDS::HistoryQosPolicy Struct Reference 898
6.85 DDS::ICopyable< T > Interface Template Reference 902
6.86 DDS::InconsistentTopicStatus Struct Reference 903
6.87 DDS::InstanceHandle_t Struct Reference 905
6.88 DDS::InstanceHandleSeq Class Reference 906
6.89 DDS::InstanceStateKind Struct Reference 907
6.90 DDS::IntSeq Class Reference 910
6.91 DDS::ITopicDescription Interface Reference 912
6.92 DDS::KeyedBytes Struct Reference 915
6.93 DDS::KeyedBytesDataReader Class Reference 918
6.94 DDS::KeyedBytesDataWriter Class Reference 920

6.95 DDS::KeyedBytesSeq Class Reference 927
6.96 DDS::KeyedBytesTypeSupport Class Reference 929
6.97 DDS::KeyedString Struct Reference 933
6.98 DDS::KeyedStringDataReader Class Reference 935
6.99 DDS::KeyedStringDataWriter Class Reference 937
6.100DDS::KeyedStringSeq Class Reference 942
6.101DDS::KeyedStringTypeSupport Class Reference 944
6.102DDS::LatencyBudgetQosPolicy Struct Reference 948
6.103DDS::LifespanQosPolicy Struct Reference 950
6.104DDS::Listener Class Reference 952
6.105DDS::LivelinessChangedStatus Struct Reference 956
6.106DDS::LivelinessLostStatus Struct Reference 958
6.107DDS::LivelinessQosPolicy Struct Reference 960
6.108DDS::LoanableSequence< E > Class Template Reference 964
6.109DDS::Locator_t Class Reference 968
6.110DDS::LocatorFilter_t Class Reference 970
6.111DDS::LocatorFilterQosPolicy Class Reference 972
6.112DDS::LocatorFilterSeq Class Reference 974
6.113DDS::LocatorSeq Class Reference 975
6.114DDS::LongDouble Struct Reference 976
6.115DDS::LongDoubleSeq Class Reference 977
6.116DDS::LongSeq Class Reference 979
6.117DDS::MultiChannelQosPolicy Class Reference 981
6.118DDS::MultiTopic Class Reference 984
6.119DDS::OfferedDeadlineMissedStatus Struct Reference 989
6.120DDS::OfferedIncompatibleQosStatus Class Reference 991
6.121DDS::OwnershipQosPolicy Struct Reference 993
6.122DDS::OwnershipStrengthQosPolicy Struct Reference 1000
6.123DDS::ParticipantBuiltinTopicData Class Reference 1002
6.124DDS::ParticipantBuiltinTopicDataDataReader Class Reference . 1005
6.125DDS::ParticipantBuiltinTopicDataSeq Class Reference 1006

6.126DDS::ParticipantBuiltinTopicDataTypeSupport Class Reference 1007
6.127DDS::PartitionQosPolicy Class Reference 1008
6.128DDS::PresentationQosPolicy Struct Reference 1012
6.129DDS::ProductVersion_t Struct Reference 1017
6.130DDS::ProfileQosPolicy Class Reference 1019
6.131DDS::Property_t Class Reference 1022
6.132DDS::PropertyQosPolicy Class Reference 1023
6.133DDS::PropertyQosPolicyHelper Class Reference 1026
6.134DDS::PropertySeq Class Reference 1027
6.135DDS::ProtocolVersion_t Struct Reference 1028
6.136DDS::PublicationBuiltinTopicData Class Reference 1030
6.137DDS::PublicationBuiltinTopicDataDataReader Class Reference . 1038
6.138DDS::PublicationBuiltinTopicDataSeq Class Reference 1039
6.139DDS::PublicationBuiltinTopicDataTypeSupport Class Reference 1040
6.140DDS::PublicationMatchedStatus Struct Reference 1041
6.141DDS::Publisher Class Reference 1044
6.142DDS::PublisherListener Class Reference 1069
6.143DDS::PublisherQos Class Reference 1074
6.144DDS::PublisherSeq Class Reference 1076
6.145DDS::PublishModeQosPolicy Class Reference 1077
6.146DDS::QosPolicyCount Struct Reference 1080
6.147DDS::QosPolicyCountSeq Class Reference 1081
6.148DDS::QueryCondition Class Reference 1082
6.149DDS::ReadCondition Class Reference 1084
6.150DDS::ReaderDataLifecycleQosPolicy Struct Reference 1087
6.151DDS::ReceiverPoolQosPolicy Class Reference 1090
6.152DDS::ReliabilityQosPolicy Struct Reference 1094
6.153DDS::ReliableReaderActivityChangedStatus Struct Reference . . 1098
6.154DDS::ReliableWriterCacheChangedStatus Struct Reference . . . 1101
6.155DDS::ReliableWriterCacheEventCount Struct Reference 1104
6.156DDS::RequestedDeadlineMissedStatus Struct Reference 1105

6.157DDS::RequestedIncompatibleQosStatus Class Reference 1107
6.158DDS::ResourceLimitsQosPolicy Struct Reference 1109
6.159DDS::Retcode_AlreadyDeleted Class Reference 1114
6.160DDS::Retcode_BadParameter Class Reference 1115
6.161DDS::Retcode_Error Class Reference 1116
6.162DDS::Retcode_IllegalOperation Class Reference 1117
6.163DDS::Retcode_ImmutablePolicy Class Reference 1118
6.164DDS::Retcode_InconsistentPolicy Class Reference 1119
6.165DDS::Retcode_NoData Class Reference 1120
6.166DDS::Retcode_NotEnabled Class Reference 1121
6.167DDS::Retcode_OutOfResources Class Reference 1122
6.168DDS::Retcode_PreconditionNotMet Class Reference 1123
6.169DDS::Retcode_Timeout Class Reference 1124
6.170DDS::Retcode_Unsupported Class Reference 1125
6.171DDS::RtpsReliableReaderProtocol_t Struct Reference 1126
6.172DDS::RtpsReliableWriterProtocol_t Struct Reference 1128
6.173DDS::RtpsWellKnownPorts_t Struct Reference 1142
6.174DDS::SampleInfo Class Reference 1148
6.175DDS::SampleInfoSeq Class Reference 1157
6.176DDS::SampleLostStatus Struct Reference 1158
6.177DDS::SampleRejectedStatus Struct Reference 1159
6.178DDS::SampleStateKind Struct Reference 1161
6.179DDS::Sequence< T > Class Template Reference 1163
6.180DDS::SequenceNumber_t Struct Reference 1175
6.181DDS::ShmemTransport Interface Reference 1177
6.182DDS::ShortSeq Class Reference 1181
6.183DDS::StatusCondition Class Reference 1183
6.184DDS::StringDataReader Class Reference 1186
6.185DDS::StringDataWriter Class Reference 1190
6.186DDS::StringSeq Class Reference 1192
6.187DDS::StringTypeSupport Class Reference 1194

6.188DDS::StructMember Class Reference 1198
6.189DDS::StructMemberSeq Class Reference 1200
6.190DDS::Subscriber Class Reference 1201
6.191DDS::SubscriberListener Class Reference 1226
6.192DDS::SubscriberQos Class Reference 1230
6.193DDS::SubscriberSeq Class Reference 1232
6.194DDS::SubscriptionBuiltinTopicData Class Reference 1233
6.195DDS::SubscriptionBuiltinTopicDataDataReader Class Reference 1241
6.196DDS::SubscriptionBuiltinTopicDataSeq Class Reference 1242
6.197DDS::SubscriptionBuiltinTopicDataTypeSupport Class Reference 1243
6.198DDS::SubscriptionMatchedStatus Struct Reference 1244
6.199DDS::SystemResourceLimitsQosPolicy Struct Reference 1247
6.200DDS::ThreadSettings_t Class Reference 1249
6.201DDS::Time_t Struct Reference 1252
6.202DDS::TimeBasedFilterQosPolicy Struct Reference 1254
6.203DDS::Topic Class Reference 1258
6.204DDS::TopicBuiltinTopicData Class Reference 1268
6.205DDS::TopicBuiltinTopicDataDataReader Class Reference 1273
6.206DDS::TopicBuiltinTopicDataSeq Class Reference 1274
6.207DDS::TopicBuiltinTopicDataTypeSupport Class Reference 1275
6.208DDS::TopicDataQosPolicy Class Reference 1276
6.209DDS::TopicListener Class Reference 1278
6.210DDS::TopicQos Class Reference 1280
6.211DDS::TransportBuiltinKindAlias Class Reference 1284
6.212DDS::TransportBuiltinQosPolicy Struct Reference 1285
6.213DDS::TransportMulticastQosPolicy Class Reference 1287
6.214DDS::TransportMulticastSettings_t Class Reference 1289
6.215DDS::TransportMulticastSettingsSeq Class Reference 1291
6.216DDS::TransportPriorityQosPolicy Struct Reference 1292
6.217DDS::TransportSelectionQosPolicy Class Reference 1294
6.218DDS::TransportUnicastQosPolicy Class Reference 1296

6.219DDS::TransportUnicastSettings_t Class Reference 1298
6.220DDS::TransportUnicastSettingsSeq Class Reference 1300
6.221DDS::TypeCode Class Reference 1301
6.222DDS::TypeCodeFactory Class Reference 1327
6.223DDS::TypedDataReader< T > Class Template Reference 1338
6.224DDS::TypedDataWriter< T > Class Template Reference 1368
6.225DDS::TypeSupport Class Reference 1385
6.226DDS::TypeSupportQosPolicy Struct Reference 1386
6.227DDS::UDPv4Transport Interface Reference 1388
6.228DDS::UDPv6Transport Interface Reference 1391
6.229DDS::UnionMember Class Reference 1394
6.230DDS::UnionMemberSeq Class Reference 1396
6.231DDS::UnsignedIntSeq Class Reference 1397
6.232DDS::UnsignedLongSeq Class Reference 1399
6.233DDS::UnsignedShortSeq Class Reference 1401
6.234DDS::UserDataQosPolicy Class Reference 1403
6.235DDS::ValueMember Class Reference 1405
6.236DDS::ValueMemberSeq Class Reference 1407
6.237DDS::VendorId_t Struct Reference 1408
6.238DDS::ViewStateKind Struct Reference 1409
6.239DDS::WaitSet Class Reference 1411
6.240DDS::WaitSetProperty_t Struct Reference 1419
6.241DDS::WcharSeq Class Reference 1421
6.242DDS::WireProtocolQosPolicy Struct Reference 1423
6.243DDS::WriterDataLifecycleQosPolicy Struct Reference 1431
6.244DDS::WstringSeq Class Reference 1434

7 Example Documentation **1437**

7.1 HelloWorld.cpp 1437
7.2 HelloWorld.idl 1442
7.3 HelloWorld_publisher.cpp 1443

7.4 HelloWorld_publisher.cs 1448
7.5 HelloWorld_subscriber.cpp 1453
7.6 HelloWorld_subscriber.cs 1458
7.7 HelloWorldPlugin.cpp 1464
7.8 HelloWorldSupport.cpp 1476

Chapter 1

RTI Data Distribution Service

Real-Time Innovations, Inc.

RTI Data Distribution Service is a data-centric communications middleware that allows developers to build high-performance distributed communications in a heterogeneous computer environment.

The Application Programming Interface (API) of RTI Data Distribution Service 4 is based on the OMG's Data Distribution Service (DDS) specification. The most recent publication of this specification can be found in the `Catalog of OMG Specifications` under "Middleware Specifications".

1.1 Feedback and Support for this Release.

For more information, visit our knowledge base, accessible from `http://www.rti.com/support`, to see sample code, general information on RTI Data Distribution Service, performance information, troubleshooting tips, and technical details.

By its very nature, the knowledge base is continuously evolving and improving. We hope that you will find it helpful. If there are questions that you would like to see addressed or comments you would like to share, please send e-mail to `support@rti.com`. We can only guarantee a response for customers with a current maintenance contract or subscription. To purchase a maintenance contract or subscription, contact your local RTI representative (see `http://www.rti.com/company/contact.html`), send an email request to `sales@rti.com`, or call +1 (408) 990-7400.

Please do not hesitate to contact RTI with questions or comments about this release. We welcome any input on how to improve RTI Data Distribution Service to suit your needs.

1.2 Available Documentation.

The documentation of this release is provided in two forms: the HTML API reference and PDF documents. If you are new to RTI Data Distribution Service 4, the **Documentation Roadmap** (p. 173) will provide direction on how to learn about this product.

1.2.1 The PDF documents 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.
- `Getting Started Guide.` 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. Developers should read this document first.
- `Getting Started Guide, Database Addendum.` Additional installation and setup information for database usage.
- `Getting Started Guide, Embedded Systems Addendum.` Additional installation and setup information for embedded systems.
- `User's Manual.` Introduction to RTI Data Distribution Service, product tour and conceptual presentation of the functionality of RTI Data Distribution Service.
- `Platform Notes.` Specific details, such as compilation setting and libraries, related to building and using RTI Data Distribution Service on the various supported platforms.
- `C API Reference Manual.` PDF version of the online HTML documentation for the C API.
- `C++ API Reference Manual.` PDF version of the online HTML documentation for the C++ API.

- `Java API Reference Manual.` PDF version of the online HTML documentation for the Java API.
- `.NET API Reference Manual.` PDF version of the online HTML documentation for the .NET API.

1.2.2 The HTML API reference contains:

- **DDS API Reference** (p. 179) - The DDS API reference.
- **RTI Data Distribution Service API Reference** (p. 192) - RTI Data Distribution Service API's independent of the DDS standard.
- **Programming How-To's** (p. 193) - Describes and shows the common tasks done using the API.
- **Programming Tools** (p. 195) - RTI Data Distribution Service helper tools.

The HTML API 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".

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

- Documentation Roadmap . . . 173
- Conventions . . . 174
- .Net Language Support . . . 177
- DDS API Reference . . . 179
  - Domain Module . . . 34
    - DomainParticipantFactory . . . 35
    - DomainParticipants . . . 37
    - Built-in Topics . . . 42
      - Participant Built-in Topics . . . 226
      - Topic Built-in Topics . . . 228
      - Publication Built-in Topics . . . 230
      - Subscription Built-in Topics . . . 232
  - Topic Module . . . 50
    - Topics . . . 51
    - User Data Type Support . . . 52
    - Type Code Support . . . 55
    - Built-in Types . . . 68
      - Octets Built-in Type . . . 222
      - KeyedOctets Built-in Type . . . 223
      - KeyedString Built-in Type . . . 224
      - String Built-in Type . . . 225
    - Dynamic Data . . . 73
  - Publication Module . . . 78
    - Publishers . . . 79
    - Data Writers . . . 82

Flow Controllers . . . 84
Subscription Module . . . 91
Subscribers . . . 94
DataReaders . . . 97
Read Conditions . . . 100
Query Conditions . . . 101
Data Samples . . . 102
Sample States . . . 103
View States . . . 104
Instance States . . . 105
Infrastructure Module . . . 107
Return Codes . . . 234
Status Kinds . . . 238
Exception Codes . . . 248
Time Support . . . 250
GUID Support . . . 254
Sequence Number Support . . . 255
QoS Policies . . . 260
USER_DATA . . . 273
TOPIC_DATA . . . 274
GROUP_DATA . . . 275
DURABILITY . . . 276
PRESENTATION . . . 279
DEADLINE . . . 281
LATENCY_BUDGET . . . 282
OWNERSHIP . . . 283
OWNERSHIP_STRENGTH . . . 285
LIVELINESS . . . 286
TIME_BASED_FILTER . . . 288
PARTITION . . . 289
RELIABILITY . . . 290
DESTINATION_ORDER . . . 292
HISTORY . . . 294
DURABILITY_SERVICE . . . 297
RESOURCE_LIMITS . . . 298
TRANSPORT_PRIORITY . . . 300
LIFESPAN . . . 301
WRITER_DATA_LIFECYCLE . . . 302
READER_DATA_LIFECYCLE . . . 303
ENTITY_FACTORY . . . 304
Extended Qos Support . . . 305
Thread Settings . . . 257
TRANSPORT_SELECTION . . . 308
TRANSPORT_UNICAST . . . 309
Unicast Settings . . . 306

TRANSPORT_MULTICAST . . . 310
Multicast Settings . . . 307
DISCOVERY . . . 320
NDDS_DISCOVERY_PEERS . . . 312
TRANSPORT_BUILTIN . . . 321
WIRE_PROTOCOL . . . 325
DATA_READER_RESOURCE_LIMITS . . . 331
DATA_WRITER_RESOURCE_LIMITS . . . 333
DATA_READER_PROTOCOL . . . 337
DATA_WRITER_PROTOCOL . . . 338
SYSTEM_RESOURCE_LIMITS . . . 339
DOMAIN_PARTICIPANT_RESOURCE_LIMITS . . . 340
EVENT . . . 341
DATABASE . . . 342
RECEIVER_POOL . . . 343
PUBLISH_MODE . . . 344
DISCOVERY_CONFIG . . . 347
TYPESUPPORT . . . 350
ASYNCHRONOUS_PUBLISHER . . . 351
EXCLUSIVE_AREA . . . 352
BATCH . . . 353
LOCATORFILTER . . . 354
MULTICHANNEL . . . 356
PROPERTY . . . 357
ENTITY_NAME . . . 364
PROFILE . . . 365
Entity Support . . . 363
Conditions and WaitSets . . . 366
Sequence Support . . . 367
Built-in Sequences . . . 109
Queries and Filters Syntax . . . 184
RTI Data Distribution Service API Reference . . . 192
Clock Selection . . . 31
Multi-channel DataWriters . . . 111
Pluggable Transports . . . 113
Using Transport Plugins . . . 119
Built-in Transport Plugins . . . 122
Configuration Utilities . . . 124
Unsupported Utilities . . . 128
Durability and Persistence . . . 129
Configuring QoS Profiles with XML . . . 135
Programming How-To's . . . 193
Publication Example . . . 138
Subscription Example . . . 139
Participant Use Cases . . . 140

Topic Use Cases 143
FlowController Use Cases 145
Publisher Use Cases 149
DataWriter Use Cases 150
Subscriber Use Cases 152
DataReader Use Cases 155
Entity Use Cases 160
Waitset Use Cases 163
Transport Use Cases 165
Filter Use Cases 166
Large Data Use Cases 171
Programming Tools 195
rtiddsgen 196
rtiddsping 208
rtiddsspy 215

Chapter 3

Class Index

3.1 Class Hierarchy

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

DDS::AllocationSettings_t . . . 369
DDS::AsynchronousPublisherQosPolicy . . . 371
DDS::BatchQosPolicy . . . 376
DDS::BuiltinTopicKey_t . . . 383
DDS::BuiltinTopicReaderResourceLimits_t . . . 385
DDS::BytesSeq . . . 397
DDS::BytesTypeSupport . . . 399
DDS::ChannelSettings_t . . . 403
DDS::Condition . . . 408
  DDS::GuardCondition . . . 892
  DDS::ReadCondition . . . 1084
    DDS::QueryCondition . . . 1082
  DDS::StatusCondition . . . 1183
NDDS::Config_LibraryVersion_t . . . 411
NDDS::ConfigLogger . . . 413
NDDS::ConfigVersion . . . 417
DDS::ContentFilterProperty_t . . . 426
DDS::DatabaseQosPolicy . . . 428
DDS::DataReaderCacheStatus . . . 460
DDS::DataReaderProtocolQosPolicy . . . 465
DDS::DataReaderProtocolStatus . . . 470
DDS::DataReaderQos . . . 480
DDS::DataReaderResourceLimitsQosPolicy . . . 486
DDS::DataWriterCacheStatus . . . 523
DDS::DataWriterProtocolQosPolicy . . . 529

DDS::DataWriterProtocolStatus 534
DDS::DataWriterQos 546
DDS::DataWriterResourceLimitsQosPolicy 552
DDS::DeadlineQosPolicy 557
DDS::DestinationOrderQosPolicy 560
DDS::DiscoveryConfigQosPolicy 563
DDS::DiscoveryQosPolicy 571
DDS::DomainParticipantFactory 649
DDS::DomainParticipantFactoryQos 673
DDS::DomainParticipantQos 683
DDS::DomainParticipantResourceLimitsQosPolicy 688
DDS::DurabilityQosPolicy 709
DDS::DurabilityServiceQosPolicy 714
DDS::Duration_t 717
DDS::DynamicDataInfo 809
DDS::DynamicDataMemberInfo 810
DDS::DynamicDataProperty_t 813
DDS::DynamicDataSeq 816
DDS::DynamicDataTypeProperty_t 818
DDS::DynamicDataTypeSerializationProperty_t 820
DDS::Entity 845
DDS::DomainEntity 576
DDS::DataReader 433
DDS::TypedDataReader< T > 1338
DDS::BytesDataReader 391
DDS::DynamicDataReader 815
DDS::KeyedBytesDataReader 918
DDS::KeyedStringDataReader 935
DDS::ParticipantBuiltinTopicDataDataReader 1005
DDS::PublicationBuiltinTopicDataDataReader 1038
DDS::StringDataReader 1186
DDS::SubscriptionBuiltinTopicDataDataReader 1241
DDS::TopicBuiltinTopicDataDataReader 1273
FooDataReader 878
FooDataReader 878
DDS::TypedDataReader< DDS::Bytes^ > 1338
DDS::TypedDataReader< DDS::DynamicData^ > 1338
DDS::TypedDataReader< DDS::KeyedBytes^ > 1338
DDS::TypedDataReader< DDS::KeyedString^ > 1338
DDS::TypedDataReader< DDS::ParticipantBuiltinTopicData^ > 1338
DDS::TypedDataReader< DDS::PublicationBuiltinTopicData^ > 1338
DDS::TypedDataReader< DDS::StringWrapper^ > 1338

DDS::TypedDataReader< DDS::SubscriptionBuiltinTopicData^ > . . . 1338
DDS::TypedDataReader< DDS::TopicBuiltinTopicData^ > . . . 1338
DDS::TypedDataReader< Foo^ > . . . 1338
DDS::DataWriter . . . 499
DDS::TypedDataWriter< T > . . . 1368
DDS::BytesDataWriter . . . 392
DDS::DynamicDataWriter . . . 828
DDS::KeyedBytesDataWriter . . . 920
DDS::KeyedStringDataWriter . . . 937
DDS::StringDataWriter . . . 1190
FooDataWriter . . . 879
FooDataWriter . . . 879
DDS::TypedDataWriter< DDS::Bytes^ > . . . 1368
DDS::TypedDataWriter< DDS::DynamicData^ > . . . 1368
DDS::TypedDataWriter< DDS::KeyedBytes^ > . . . 1368
DDS::TypedDataWriter< DDS::KeyedString^ > . . . 1368
DDS::TypedDataWriter< DDS::StringWrapper^ > . . . 1368
DDS::TypedDataWriter< Foo^ > . . . 1368
DDS::Publisher . . . 1044
DDS::Subscriber . . . 1201
DDS::Topic . . . 1258
DDS::DomainParticipant . . . 577
DDS::EntityFactoryQosPolicy . . . 851
DDS::EntityNameQosPolicy . . . 854
DDS::EnumMember . . . 856
DDS::EventQosPolicy . . . 858
DDS::Exception . . . 861
DDS::Retcode_AlreadyDeleted . . . 1114
DDS::Retcode_BadParameter . . . 1115
DDS::Retcode_Error . . . 1116
DDS::Retcode_IllegalOperation . . . 1117
DDS::Retcode_ImmutablePolicy . . . 1118
DDS::Retcode_InconsistentPolicy . . . 1119
DDS::Retcode_NoData . . . 1120
DDS::Retcode_NotEnabled . . . 1121
DDS::Retcode_OutOfResources . . . 1122
DDS::Retcode_PreconditionNotMet . . . 1123
DDS::Retcode_Timeout . . . 1124
DDS::Retcode_Unsupported . . . 1125
DDS::ExclusiveAreaQosPolicy . . . 862
DDS::FlowController . . . 867
DDS::FlowControllerProperty_t . . . 871
DDS::FlowControllerTokenBucketProperty_t . . . 873
DDS::GroupDataQosPolicy . . . 890

DDS::GUID_t . . . 894
DDS::HistoryQosPolicy . . . 898
DDS::ICopyable< T > . . . 902
DDS::Bytes . . . 388
DDS::DynamicData . . . 719
DDS::KeyedBytes . . . 915
DDS::KeyedString . . . 933
DDS::ParticipantBuiltinTopicData . . . 1002
DDS::PublicationBuiltinTopicData . . . 1030
DDS::SubscriptionBuiltinTopicData . . . 1233
DDS::TopicBuiltinTopicData . . . 1268
Foo . . . 877
Foo . . . 877
DDS::ICopyable< DDS::Bytes^ > . . . 902
DDS::ICopyable< DDS::DynamicData^ > . . . 902
DDS::ICopyable< DDS::KeyedBytes^ > . . . 902
DDS::ICopyable< DDS::KeyedString^ > . . . 902
DDS::ICopyable< DDS::ParticipantBuiltinTopicData^ > . . . 902
DDS::ICopyable< DDS::PublicationBuiltinTopicData^ > . . . 902
DDS::ICopyable< DDS::StringWrapper^ > . . . 902
DDS::ICopyable< DDS::SubscriptionBuiltinTopicData^ > . . . 902
DDS::ICopyable< DDS::TopicBuiltinTopicData^ > . . . 902
DDS::ICopyable< Foo^ > . . . 902
DDS::InconsistentTopicStatus . . . 903
DDS::InstanceHandle_t . . . 905
DDS::InstanceStateKind . . . 907
DDS::ITopicDescription . . . 912
DDS::ContentFilteredTopic . . . 419
DDS::MultiTopic . . . 984
DDS::Topic . . . 1258
DDS::KeyedBytesSeq . . . 927
DDS::KeyedBytesTypeSupport . . . 929
DDS::KeyedStringSeq . . . 942
DDS::KeyedStringTypeSupport . . . 944
DDS::LatencyBudgetQosPolicy . . . 948
DDS::LifespanQosPolicy . . . 950
DDS::Listener . . . 952
DDS::DataReaderListener . . . 461
DDS::SubscriberListener . . . 1226
DDS::DomainParticipantListener . . . 675
DDS::DataWriterListener . . . 524
DDS::PublisherListener . . . 1069
DDS::DomainParticipantListener . . . 675
DDS::TopicListener . . . 1278
DDS::DomainParticipantListener . . . 675

DDS::LivelinessChangedStatus 956
DDS::LivelinessLostStatus 958
DDS::LivelinessQosPolicy 960
DDS::Locator_t 968
DDS::LocatorFilter_t 970
DDS::LocatorFilterQosPolicy 972
DDS::LongDouble 976
DDS::MultiChannelQosPolicy 981
DDS::OfferedDeadlineMissedStatus 989
DDS::OfferedIncompatibleQosStatus 991
DDS::OwnershipQosPolicy 993
DDS::OwnershipStrengthQosPolicy 1000
DDS::ParticipantBuiltinTopicDataSeq 1006
DDS::ParticipantBuiltinTopicDataTypeSupport 1007
DDS::PartitionQosPolicy 1008
DDS::PresentationQosPolicy 1012
DDS::ProductVersion_t 1017
DDS::ProfileQosPolicy 1019
DDS::Property_t 1022
DDS::PropertyQosPolicy 1023
DDS::PropertyQosPolicyHelper 1026
DDS::ProtocolVersion_t 1028
DDS::PublicationBuiltinTopicDataSeq 1039
DDS::PublicationBuiltinTopicDataTypeSupport 1040
DDS::PublicationMatchedStatus 1041
DDS::PublisherQos 1074
DDS::PublishModeQosPolicy 1077
DDS::QosPolicyCount 1080
DDS::ReaderDataLifecycleQosPolicy 1087
DDS::ReceiverPoolQosPolicy 1090
DDS::ReliabilityQosPolicy 1094
DDS::ReliableReaderActivityChangedStatus 1098
DDS::ReliableWriterCacheChangedStatus 1101
DDS::ReliableWriterCacheEventCount 1104
DDS::RequestedDeadlineMissedStatus 1105
DDS::RequestedIncompatibleQosStatus 1107
DDS::ResourceLimitsQosPolicy 1109
DDS::RtpsReliableReaderProtocol_t 1126
DDS::RtpsReliableWriterProtocol_t 1128
DDS::RtpsWellKnownPorts_t 1142
DDS::SampleInfo 1148
DDS::SampleLostStatus 1158
DDS::SampleRejectedStatus 1159
DDS::SampleStateKind 1161
DDS::Sequence< T > 1163

DDS::ChannelSettingsSeq 405
DDS::ConditionSeq 410
DDS::DataReaderSeq 498
DDS::EnumMemberSeq 857
DDS::InstanceHandleSeq 906
DDS::LocatorFilterSeq 974
DDS::LocatorSeq 975
DDS::LongDoubleSeq 977
DDS::PropertySeq 1027
DDS::PublisherSeq 1076
DDS::QosPolicyCountSeq 1081
DDS::StructMemberSeq 1200
DDS::SubscriberSeq 1232
DDS::TransportMulticastSettingsSeq 1291
DDS::TransportUnicastSettingsSeq 1300
DDS::UnionMemberSeq 1396
DDS::ValueMemberSeq 1407
DDS::WstringSeq 1434
DDS::Sequence< DDS::ChannelSettings_t^ > 1163
DDS::Sequence< DDS::Condition^ > 1163
DDS::Sequence< DDS::DataReader^ > 1163
DDS::Sequence< DDS::DataWriter^ > 1163
DDS::Sequence< DDS::Discovery_EndpointInformation^ > 1163
DDS::Sequence< DDS::Discovery_ParticipantInformation^ > 1163
DDS::Sequence< DDS::EnumMember^ > 1163
DDS::Sequence< DDS::InstanceHandle_t > 1163
DDS::Sequence< DDS::Locator_t^ > 1163
DDS::Sequence< DDS::LocatorFilter_t^ > 1163
DDS::Sequence< DDS::LongDouble > 1163
DDS::Sequence< DDS::ParticipantBuiltinTopicData^ > 1163
DDS::LoanableSequence< DDS::ParticipantBuiltinTopicData^ > . 964
DDS::Sequence< DDS::Property_t^ > 1163
DDS::Sequence< DDS::PublicationBuiltinTopicData^ > 1163
DDS::LoanableSequence< DDS::PublicationBuiltinTopicData^ > . 964
DDS::Sequence< DDS::Publisher^ > 1163
DDS::Sequence< DDS::QosPolicyCount > 1163
DDS::Sequence< DDS::SampleInfo^ > 1163
DDS::LoanableSequence< DDS::SampleInfo^ > 964
DDS::Sequence< DDS::StructMember^ > 1163
DDS::Sequence< DDS::Subscriber^ > 1163
DDS::Sequence< DDS::SubscriptionBuiltinTopicData^ > 1163
DDS::LoanableSequence< DDS::SubscriptionBuiltinTopicData^ > 964
DDS::Sequence< DDS::TopicBuiltinTopicData^ > 1163
DDS::LoanableSequence< DDS::TopicBuiltinTopicData^ > 964

DDS::Sequence< DDS::TransportEncapsulationSettings_t^ > 1163
DDS::Sequence< DDS::TransportMulticastSettings_t^ > 1163
DDS::Sequence< DDS::TransportUnicastSettings_t^ > 1163
DDS::Sequence< DDS::UnionMember^ > 1163
DDS::Sequence< DDS::ValueMember^ > 1163
DDS::Sequence< E > 1163
DDS::LoanableSequence< E > 964
DDS::SampleInfoSeq 1157
FooSeq 880
DDS::Sequence< Foo^ > 1163
DDS::LoanableSequence< Foo^ > 964
DDS::Sequence< System::Boolean > 1163
DDS::BooleanSeq 381
DDS::Sequence< System::Byte > 1163
DDS::ByteSeq 395
DDS::Sequence< System::Char > 1163
DDS::CharSeq 406
DDS::WcharSeq 1421
DDS::Sequence< System::Double > 1163
DDS::DoubleSeq 707
DDS::Sequence< System::Int16 > 1163
DDS::ShortSeq 1181
DDS::Sequence< System::Int32 > 1163
DDS::IntSeq 910
DDS::Sequence< System::Int64 > 1163
DDS::LongSeq 979
DDS::Sequence< System::Single > 1163
DDS::FloatSeq 865
DDS::Sequence< System::String^ > 1163
DDS::Sequence< System::UInt16 > 1163
DDS::UnsignedShortSeq 1401
DDS::Sequence< System::UInt32 > 1163
DDS::UnsignedIntSeq 1397
DDS::Sequence< System::UInt64 > 1163
DDS::UnsignedLongSeq 1399
DDS::SequenceNumber_t 1175
DDS::ShmemTransport 1177
DDS::StringSeq 1192
DDS::StringTypeSupport 1194
DDS::StructMember 1198
DDS::SubscriberQos 1230
DDS::SubscriptionBuiltinTopicDataSeq 1242

DDS::SubscriptionBuiltinTopicDataTypeSupport 1243
DDS::SubscriptionMatchedStatus 1244
DDS::SystemResourceLimitsQosPolicy 1247
DDS::ThreadSettings_t 1249
DDS::Time_t 1252
DDS::TimeBasedFilterQosPolicy 1254
DDS::TopicBuiltinTopicDataSeq 1274
DDS::TopicBuiltinTopicDataTypeSupport 1275
DDS::TopicDataQosPolicy 1276
DDS::TopicQos 1280
DDS::TransportBuiltinKindAlias 1284
DDS::TransportBuiltinQosPolicy 1285
DDS::TransportMulticastQosPolicy 1287
DDS::TransportMulticastSettings_t 1289
DDS::TransportPriorityQosPolicy 1292
DDS::TransportSelectionQosPolicy 1294
DDS::TransportUnicastQosPolicy 1296
DDS::TransportUnicastSettings_t 1298
DDS::TypeCode 1301
DDS::TypeCodeFactory 1327
DDS::TypeSupport 1385
DDS::DynamicDataTypeSupport 822
FooTypeSupport 884
DDS::TypeSupportQosPolicy 1386
DDS::UDPv4Transport 1388
DDS::UDPv6Transport 1391
DDS::UnionMember 1394
DDS::UserDataQosPolicy 1403
DDS::ValueMember 1405
DDS::VendorId_t 1408
DDS::ViewStateKind 1409
DDS::WaitSet 1411
DDS::WaitSetProperty_t 1419
DDS::WireProtocolQosPolicy 1423
DDS::WriterDataLifecycleQosPolicy 1431

Chapter 4

Class Index

4.1 Class List

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

DDS::AllocationSettings_t (Resource allocation settings) 369
DDS::AsynchronousPublisherQosPolicy (Configures the mechanism that sends user data in an external middleware thread) 371
DDS::BatchQosPolicy (Used to configure batching of multiple samples into a single network packet in order to increase throughput for small samples) 376
DDS::BooleanSeq (Instantiates `DDS::Sequence` (p. 1163) < System::Boolean >) 381
DDS::BuiltinTopicKey_t (The key type of the built-in topic types) 383
DDS::BuiltinTopicReaderResourceLimits_t (Built-in topic reader's resource limits) 385
DDS::Bytes (Built-in type consisting of a variable-length array of opaque bytes) 388
DDS::BytesDataReader (<<*interface*>> *(p. 175)* Instantiates `DataReader` (p. 433) < **DDS::Bytes** (p. 388) >) 391
DDS::BytesDataWriter (<<*interface*>> *(p. 175)* Instantiates `DataWriter` (p. 499) < **DDS::Bytes** (p. 388) >) 392
DDS::ByteSeq (Instantiates `DDS::Sequence` (p. 1163) < System::Byte >) 395
DDS::BytesSeq (Instantiates `DDS::Sequence` (p. 1163) < **DDS::Bytes** (p. 388) >) 397
DDS::BytesTypeSupport (<<*interface*>> *(p. 175)* **DDS::Bytes** (p. 388) type support) 399

DDS::ChannelSettings_t (Type used to configure the properties of a channel) . . . 403
DDS::ChannelSettingsSeq (Declares IDL `sequence<` **DDS::ChannelSettings_t** (p. 403) >) . . . 405
DDS::CharSeq (Instantiates `DDS::Sequence` (p. 1163) < System::Char >) . . . 406
DDS::Condition (<<*interface*>> *(p. 175)* Root class for all the conditions that may be attached to a **DDS::WaitSet** (p. 1411)) . . . 408
DDS::ConditionSeq (Instantiates `DDS::Sequence` (p. 1163) < **DDS::Condition** (p. 408) >) . . . 410
NDDS::Config_LibraryVersion_t (The version of a single library shipped as part of an RTI Data Distribution Service distribution) . . . 411
NDDS::ConfigLogger (<<*interface*>> *(p. 175)* The singleton type used to configure RTI Data Distribution Service logging) 413
NDDS::ConfigVersion (<<*interface*>> *(p. 175)* The version of an RTI Data Distribution Service distribution) . . . 417
DDS::ContentFilteredTopic (<<*interface*>> *(p. 175)* Specialization of DDS::TopicDescription that allows for content-based subscriptions) . . . 419
DDS::ContentFilterProperty_t (<<*eXtension*>> *(p. 174)* Type used to provide all the required information to enable content filtering) . . . 426
DDS::DatabaseQosPolicy (Various threads and resource limits settings used by RTI Data Distribution Service to control its internal database) . . . 428
DDS::DataReader (<<*interface*>> *(p. 175)* 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. 1201)) . . . 433
DDS::DataReaderCacheStatus (<<*eXtension*>> *(p. 174)* The status of the reader's cache) . . . 460
DDS::DataReaderListener (<<*interface*>> *(p. 175)* **DDS::Listener** (p. 952) for reader status) . . . 461
DDS::DataReaderProtocolQosPolicy (Along with **DDS::WireProtocolQosPolicy** (p. 1423) and **DDS::DataWriterProtocolQosPolicy** (p. 529), this QoS policy configures the DDS on-the-network protocol (RTPS)) . . . 465
DDS::DataReaderProtocolStatus (<<*eXtension*>> *(p. 174)* The status of a reader's internal protocol related metrics, like the number of samples received, filtered, rejected; and status of wire protocol traffic) . . . 470
DDS::DataReaderQos (QoS policies supported by a **DDS::DataReader** (p. 433) entity) . . . 480

DDS::DataReaderResourceLimitsQosPolicy (Various settings that configure how a **DDS::DataReader** (p. 433) allocates and uses physical memory for internal resources) 486
DDS::DataReaderSeq (Declares IDL `sequence` < **DDS::DataReader** (p. 433) >) 498
DDS::DataWriter (<<*interface*>> (p. *175*) Allows an application to set the value of the data to be published under a given **DDS::Topic** (p. 1258)) 499
DDS::DataWriterCacheStatus (<<*eXtension*>> (p. *174*) The status of the writer's cache) 523
DDS::DataWriterListener (<<*interface*>> (p. *175*) **DDS::Listener** (p. 952) for writer status) 524
DDS::DataWriterProtocolQosPolicy (Protocol that applies only to **DDS::DataWriter** (p. 499) instances) 529
DDS::DataWriterProtocolStatus (<<*eXtension*>> (p. *174*) The status of a writer's internal protocol related metrics, like the number of samples pushed, pulled, filtered; and status of wire protocol traffic) 534
DDS::DataWriterQos (QoS policies supported by a **DDS::DataWriter** (p. 499) entity) 546
DDS::DataWriterResourceLimitsQosPolicy (Various settings that configure how a **DDS::DataWriter** (p. 499) allocates and uses physical memory for internal resources) 552
DDS::DeadlineQosPolicy (Expresses the maximum duration (deadline) within which an instance is expected to be updated) . . 557
DDS::DestinationOrderQosPolicy (Controls how the middleware will deal with data sent by multiple **DDS::DataWriter** (p. 499) entities for the same instance of data (i.e., same **DDS::Topic** (p. 1258) and key)) 560
DDS::DiscoveryConfigQosPolicy (Settings for discovery configuration) 563
DDS::DiscoveryQosPolicy (Configures the mechanism used by the middleware to automatically discover and connect with new remote applications) 571
DDS::DomainEntity (<<*interface*>> (p. *175*) Abstract base class for all DDS entities except for the **DDS::DomainParticipant** (p. 577)) 576
DDS::DomainParticipant (<<*interface*>> (p. *175*) Container for all **DDS::DomainEntity** (p. 576) objects) 577
DDS::DomainParticipantFactory (<<*singleton*>> (p. *175*) <<*interface*>> (p. *175*) Allows creation and destruction of **DDS::DomainParticipant** (p. 577) objects) 649
DDS::DomainParticipantFactoryQos (QoS policies supported by a **DDS::DomainParticipantFactory** (p. 649)) 673
DDS::DomainParticipantListener (<<*interface*>> (p. *175*) **Listener** (p. 952) for participant status) 675

DDS::DomainParticipantQos (QoS policies supported by a **DDS::DomainParticipant** (p. 577) entity) 683
DDS::DomainParticipantResourceLimitsQosPolicy (Various settings that configure how a **DDS::DomainParticipant** (p. 577) allocates and uses physical memory for internal resources, including the maximum sizes of various properties) 688
DDS::DoubleSeq (Instantiates `DDS::Sequence` (p. 1163) < System::Double >) 707
DDS::DurabilityQosPolicy (This QoS policy specifies whether or not RTI Data Distribution Service will store and deliver previously published data samples to new **DDS::DataReader** (p. 433) entities that join the network later) 709
DDS::DurabilityServiceQosPolicy (Various settings to configure the external *RTI Persistence Service* used by RTI Data Distribution Service for DataWriters with a **DDS::DurabilityQosPolicy** (p. 709) setting of DDS::DurabilityQosPolicyKind::PERSISTENT_DURABILITY_QOS or DDS::DurabilityQosPolicyKind::TRANSIENT_DURABILITY_QOS) 714
DDS::Duration_t (Type for *duration* representation) 717
DDS::DynamicData (A sample of any complex data type, which can be inspected and manipulated reflectively) 719
DDS::DynamicDataInfo (A descriptor for a **DDS::DynamicData** (p. 719) object) 809
DDS::DynamicDataMemberInfo (A descriptor for a single member (i.e. field) of dynamically defined data type) 810
DDS::DynamicDataProperty_t (A collection of attributes used to configure **DDS::DynamicData** (p. 719) objects) 813
DDS::DynamicDataReader (Reads (subscribes to) objects of type **DDS::DynamicData** (p. 719)) 815
DDS::DynamicDataSeq (An ordered collection of **DDS::DynamicData** (p. 719) elements) 816
DDS::DynamicDataTypeProperty_t (A collection of attributes used to configure **DDS::DynamicDataTypeSupport** (p. 822) objects) 818
DDS::DynamicDataTypeSerializationProperty_t (Properties that govern how data of a certain type will be serialized on the network) 820
DDS::DynamicDataTypeSupport (A factory for registering a dynamically defined type and creating **DDS::DynamicData** (p. 719) objects) 822
DDS::DynamicDataWriter (Writes (publishes) objects of type **DDS::DynamicData** (p. 719)) 828
DDS::Entity (<<***interface***>> *(p. 175)* Abstract base class for all the DDS objects that support QoS policies, a listener, and a status condition) 845

DDS::EntityFactoryQosPolicy (A QoS policy for all **DDS::Entity** (p. 845) types that can act as factories for one or more other **DDS::Entity** (p. 845) types) 851
DDS::EntityNameQosPolicy (Assigns a name to a **DDS::DomainParticipant** (p. 577). This name will be visible during the discovery process and in RTI tools to help you visualize and debug your system) 854
DDS::EnumMember (A description of a member of an enumeration) 856
DDS::EnumMemberSeq (Defines a sequence of enumerator members) 857
DDS::EventQosPolicy (Settings for event) 858
DDS::Exception (Superclass of all exceptions thrown by the RTI Data Distribution Service API) 861
DDS::ExclusiveAreaQosPolicy (Configures multi-thread concurrency and deadlock prevention capabilities) 862
DDS::FloatSeq (Instantiates `DDS::Sequence` (p. 1163) < System::Single >) 865
DDS::FlowController (<<*interface*>> *(p. 175)* A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous **DDS::DataWriter** (p. 499) instances are allowed to write data) 867
DDS::FlowControllerProperty_t (Determines the flow control characteristics of the **DDS::FlowController** (p. 867)) . . . 871
DDS::FlowControllerTokenBucketProperty_t (**DDS::FlowController** (p. 867) uses the popular token bucket approach for open loop network flow control. The flow control characteristics are determined by the token bucket properties) 873
Foo (A representative user-defined data type) 877
FooDataReader (<<*interface*>> *(p. 175)* <<*generic*>> *(p. 175)* User data type-specific data reader) 878
FooDataWriter (<<*interface*>> *(p. 175)* <<*generic*>> *(p. 175)* User data type specific data writer) 879
FooSeq (<<*interface*>> *(p. 175)* <<*generic*>> *(p. 175)* A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as `Foo` (p. 877)) 880
FooTypeSupport (<<*interface*>> *(p. 175)* <<*generic*>> *(p. 175)* User data type specific interface) 884
DDS::GroupDataQosPolicy (Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 42) during discovery) 890
DDS::GuardCondition (<<*interface*>> *(p. 175)* A specific **DDS::Condition** (p. 408) whose `trigger_value` is completely under the control of the application) 892

DDS::GUID_t (Type for *GUID* (Global Unique Identifier) representation) 894
DDS::HistoryQosPolicy (Specifies the behavior of RTI Data Distribution Service 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) 898
DDS::ICopyable< T > (<<*eXtension*>> *(p. 174)* <<*interface*>> *(p. 175)* Interface for all the user-defined data type classes that support copy) 902
DDS::InconsistentTopicStatus (DDS::StatusKind::INCONSISTENT_TOPIC_STATUS) 903
DDS::InstanceHandle_t (Type definition for an instance handle) . 905
DDS::InstanceHandleSeq (Instantiates `DDS::Sequence` (p. 1163) < **DDS::InstanceHandle_t** (p. 905) >) 906
DDS::InstanceStateKind (Indicates is the samples are from a live **DDS::DataWriter** (p. 499) or not) 907
DDS::IntSeq (Instantiates `DDS::Sequence` (p. 1163) < System::Int32 >) 910
DDS::ITopicDescription (<<*interface*>> *(p. 175)* Base class for **DDS::Topic** (p. 1258), **DDS::ContentFilteredTopic** (p. 419), and **DDS::MultiTopic** (p. 984)) 912
DDS::KeyedBytes (Built-in type consisting of a variable-length array of opaque bytes and a string that is the key) 915
DDS::KeyedBytesDataReader (<<*interface*>> *(p. 175)* Instantiates `DataReader` (p. 433) < **DDS::KeyedBytes** (p. 915) >) 918
DDS::KeyedBytesDataWriter (<<*interface*>> *(p. 175)* Instantiates `DataWriter` (p. 499) < **DDS::KeyedBytes** (p. 915) >) 920
DDS::KeyedBytesSeq (Instantiates `DDS::Sequence` (p. 1163) < **DDS::KeyedBytes** (p. 915) >) 927
DDS::KeyedBytesTypeSupport (<<*interface*>> *(p. 175)* **DDS::KeyedBytes** (p. 915) type support) 929
DDS::KeyedString (Keyed string built-in type) 933
DDS::KeyedStringDataReader (<<*interface*>> *(p. 175)* Instantiates `DataReader` (p. 433) < **DDS::KeyedString** (p. 933) >) 935
DDS::KeyedStringDataWriter (<<*interface*>> *(p. 175)* Instantiates `DataWriter` (p. 499) < **DDS::KeyedString** (p. 933) >) 937
DDS::KeyedStringSeq (Instantiates `DDS::Sequence` (p. 1163) < **DDS::KeyedString** (p. 933) >) 942
DDS::KeyedStringTypeSupport (<<*interface*>> *(p. 175)* Keyed string type support) 944

DDS::LatencyBudgetQosPolicy (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) 948
DDS::LifespanQosPolicy (Specifies how long the data written by the **DDS::DataWriter** (p. 499) is considered valid) 950
DDS::Listener (<<*interface*>> *(p. 175)* Abstract base class for all **Listener** (p. 952) interfaces) 952
DDS::LivelinessChangedStatus (DDS::StatusKind::LIVELINESS_CHANGED_STATUS) 956
DDS::LivelinessLostStatus (DDS::StatusKind::LIVELINESS_LOST_STATUS) 958
DDS::LivelinessQosPolicy (Specifies and configures the mechanism that allows **DDS::DataReader** (p. 433) entities to detect when **DDS::DataWriter** (p. 499) entities become disconnected or "dead.") 960
DDS::LoanableSequence< E > (A sequence implementation used internally by the middleware to efficiently manage memory during **DDS::TypedDataReader::read** (p. 1341) and **DDS::TypedDataReader::take** (p. 1342) operations) . . . 964
DDS::Locator_t (<<*eXtension*>> *(p. 174)* Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports) . . . 968
DDS::LocatorFilter_t (The QoS policy used to report the configuration of a MultiChannel **DataWriter** (p. 499) as part of **DDS::PublicationBuiltinTopicData** (p. 1030)) 970
DDS::LocatorFilterQosPolicy (The QoS policy used to report the configuration of a MultiChannel **DataWriter** (p. 499) as part of **DDS::PublicationBuiltinTopicData** (p. 1030)) 972
DDS::LocatorFilterSeq (Declares IDL `sequence<` **DDS::LocatorFilter_t** (p. 970) >) 974
DDS::LocatorSeq (Declares IDL `sequence <` **DDS::Locator_t** (p. 968) >) 975
DDS::LongDouble (Defines an extra-precision floating-point data type, equivalent to IDL/CDR `long double`) 976
DDS::LongDoubleSeq (Instantiates `DDS::Sequence` (p. 1163) < **DDS::LongDouble** (p. 976) >) 977
DDS::LongSeq (Instantiates `DDS::Sequence` (p. 1163) < System::Int64 >) 979
DDS::MultiChannelQosPolicy (Configures the ability of a **DataWriter** (p. 499) to send data on different multicast groups (addresses) based on the value of the data) 981
DDS::MultiTopic ([**Not supported (optional)**] <<*interface*>> *(p. 175)* A specialization of DDS::TopicDescription that allows subscriptions that combine/filter/rearrange data coming from several topics) 984

DDS::OfferedDeadlineMissedStatus (DDS::StatusKind::OFFERED_DEADLINE_MISSED_STATUS) 989
DDS::OfferedIncompatibleQosStatus (DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS) 991
DDS::OwnershipQosPolicy (Specifies whether it is allowed for multiple **DDS::DataWriter** (p. 499) (s) to write the same instance of the data and if so, how these modifications should be arbitrated) 993
DDS::OwnershipStrengthQosPolicy (Specifies the value of the strength used to arbitrate among multiple **DDS::DataWriter** (p. 499) objects that attempt to modify the same instance of a data type (identified by **DDS::Topic** (p. 1258) + key)) 1000
DDS::ParticipantBuiltinTopicData (Entry created when a **DomainParticipant** (p. 577) object is discovered) 1002
DDS::ParticipantBuiltinTopicDataDataReader (Instantiates `DataReader (p. 433)` < **DDS::ParticipantBuiltinTopicData** (p. 1002) >) 1005
DDS::ParticipantBuiltinTopicDataSeq (Instantiates `DDS::Sequence (p. 1163)` < **DDS::ParticipantBuiltinTopicData** (p. 1002) >) 1006
DDS::ParticipantBuiltinTopicDataTypeSupport (Instantiates `TypeSupport (p. 1385)` < **DDS::ParticipantBuiltinTopicData** (p. 1002) >) 1007
DDS::PartitionQosPolicy (Set of strings that introduces a logical partition among the topics visible by a **DDS::Publisher** (p. 1044) and a **DDS::Subscriber** (p. 1201)) 1008
DDS::PresentationQosPolicy (Specifies how the samples representing changes to data instances are presented to a subscribing application) 1012
DDS::ProductVersion_t (<<***eXtension***>> *(p. 174)* Type used to represent the current version of RTI Data Distribution Service) 1017
DDS::ProfileQosPolicy (Configures the way that XML documents containing QoS profiles are loaded by RTI Data Distribution Service) 1019
DDS::Property_t (Properties are name/value pairs objects) 1022
DDS::PropertyQosPolicy (Stores name/value(string) pairs that can be used to configure certain parameters of RTI Data Distribution Service 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) 1023
DDS::PropertyQosPolicyHelper (Policy Helpers which facilitate management of the properties in the input policy) 1026

DDS::PropertySeq (Declares IDL `sequence` < **DDS::Property_t** (p. 1022) >) 1027
DDS::ProtocolVersion_t (<<*eXtension*>> *(p. 174)* Type used to represent the version of the RTPS protocol) 1028
DDS::PublicationBuiltinTopicData (Entry created when a **DDS::DataWriter** (p. 499) is discovered in association with its **Publisher** (p. 1044)) 1030
DDS::PublicationBuiltinTopicDataDataReader (Instantiates `DataReader` (p. 433) < **DDS::PublicationBuiltinTopicData** (p. 1030) >) 1038
DDS::PublicationBuiltinTopicDataSeq (Instantiates `DDS::Sequence` (p. 1163) < **DDS::PublicationBuiltinTopicData** (p. 1030) >) 1039
DDS::PublicationBuiltinTopicDataTypeSupport (Instantiates `TypeSupport` (p. 1385) < **DDS::PublicationBuiltinTopicData** (p. 1030) >) 1040
DDS::PublicationMatchedStatus (DDS::StatusKind::PUBLICATION_MATCHED_STATUS) 1041
DDS::Publisher (<<*interface*>> *(p. 175)* A publisher is the object responsible for the actual dissemination of publications) . . . 1044
DDS::PublisherListener (<<*interface*>> *(p. 175)* **DDS::Listener** (p. 952) for **DDS::Publisher** (p. 1044) status) 1069
DDS::PublisherQos (QoS policies supported by a **DDS::Publisher** (p. 1044) entity) 1074
DDS::PublisherSeq (Declares IDL `sequence` < **DDS::Publisher** (p. 1044) >) 1076
DDS::PublishModeQosPolicy (Specifies how RTI Data Distribution Service sends application data on the network. This QoS policy can be used to tell RTI Data Distribution Service to use its *own* thread to send data, instead of the user thread) . 1077
DDS::QosPolicyCount (Type to hold a counter for a DDS::QosPolicyId_t) 1080
DDS::QosPolicyCountSeq (Declares IDL `sequence` < **DDS::QosPolicyCount** (p. 1080) >) 1081
DDS::QueryCondition (<<*interface*>> *(p. 175)* These are specialised **DDS::ReadCondition** (p. 1084) objects that allow the application to also specify a filter on the locally available data) 1082
DDS::ReadCondition (<<*interface*>> *(p. 175)* Conditions specifically dedicated to read operations and attached to one **DDS::DataReader** (p. 433)) 1084
DDS::ReaderDataLifecycleQosPolicy (Controls how a **DataReader** (p. 433) manages the lifecycle of the data that it has received) 1087

DDS::ReceiverPoolQosPolicy (Configures threads used by RTI Data Distribution Service to receive and process data from transports (for example, UDP sockets)) 1090
DDS::ReliabilityQosPolicy (Indicates the level of reliability offered/requested by RTI Data Distribution Service) 1094
DDS::ReliableReaderActivityChangedStatus (<<***eXtension***>> *(p. 174)* Describes the activity (i.e. are acknowledgements forthcoming) of reliable readers matched to a reliable writer) 1098
DDS::ReliableWriterCacheChangedStatus (<<***eXtension***>> *(p. 174)* A summary of the state of a data writer's cache of unacknowledged samples written) 1101
DDS::ReliableWriterCacheEventCount (<<***eXtension***>> *(p. 174)* The number of times the number of unacknowledged samples in the cache of a reliable writer hit a certain well-defined threshold) 1104
DDS::RequestedDeadlineMissedStatus (DDS::StatusKind::REQUESTED_DEADLINE_MISSED_STATUS) 1105
DDS::RequestedIncompatibleQosStatus (DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS) 1107
DDS::ResourceLimitsQosPolicy (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) 1109
DDS::Retcode_AlreadyDeleted (The object target of this operation has already been deleted) 1114
DDS::Retcode_BadParameter (Illegal parameter value) 1115
DDS::Retcode_Error (Generic, unspecified error) 1116
DDS::Retcode_IllegalOperation (The operation was called under improper circumstances) 1117
DDS::Retcode_ImmutablePolicy (Application attempted to modify an immutable QoS policy) 1118
DDS::Retcode_InconsistentPolicy (Application specified a set of QoS policies that are not consistent with each other) 1119
DDS::Retcode_NoData (Indicates a transient situation where the operation did not return any data but there is no inherent error) 1120
DDS::Retcode_NotEnabled (Operation invoked on a **DDS::Entity** (p. 845) that is not yet enabled) 1121
DDS::Retcode_OutOfResources (RTI Data Distribution Service ran out of the resources needed to complete the operation) 1122
DDS::Retcode_PreconditionNotMet (A pre-condition for the operation was not met) 1123

DDS::Retcode_Timeout (The operation timed out) 1124
DDS::Retcode_Unsupported (Unsupported operation. Can only returned by operations that are unsupported) 1125
DDS::RtpsReliableReaderProtocol_t (Qos related to reliable reader protocol defined in RTPS) 1126
DDS::RtpsReliableWriterProtocol_t (QoS related to the reliable writer protocol defined in RTPS) 1128
DDS::RtpsWellKnownPorts_t (RTPS well-known port mapping configuration) 1142
DDS::SampleInfo (Information that accompanies each sample that is `read` or `taken`) 1148
DDS::SampleInfoSeq (Declares IDL `sequence` < **DDS::SampleInfo** (p. 1148) >) 1157
DDS::SampleLostStatus (DDS::StatusKind::SAMPLE_LOST_STATUS) 1158
DDS::SampleRejectedStatus (DDS::StatusKind::SAMPLE_REJECTED_STATUS) 1159
DDS::SampleStateKind (Indicates whether or not a sample has ever been read) 1161
DDS::Sequence< T > (<<***interface***>> *(p. 175)* <<***generic***>> *(p. 175)* A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as `Foo` (p. 877)) 1163
DDS::SequenceNumber_t (Type for *sequence* number representation) 1175
DDS::ShmemTransport (Built-in transport plug-in for inter-process communications using shared memory) 1177
DDS::ShortSeq (Instantiates `DDS::Sequence` (p. 1163) < System::Int16 >) 1181
DDS::StatusCondition (<<***interface***>> *(p. 175)* A specific **DDS::Condition** (p. 408) that is associated with each **DDS::Entity** (p. 845)) 1183
DDS::StringDataReader (<<***interface***>> *(p. 175)* Instantiates `DataReader` (p. 433) < System::String >) 1186
DDS::StringDataWriter (<<***interface***>> *(p. 175)* Instantiates `DataWriter` (p. 499) < System::String >) 1190
DDS::StringSeq (Instantiates `DDS::Sequence` (p. 1163) < System::String > with value type semantics) 1192
DDS::StringTypeSupport (<<***interface***>> *(p. 175)* String type support) 1194
DDS::StructMember (A description of a member of a struct) . . . 1198
DDS::StructMemberSeq (Defines a sequence of struct members) . 1200
DDS::Subscriber (<<***interface***>> *(p. 175)* A subscriber is the object responsible for actually receiving data from a subscription) 1201

DDS::SubscriberListener (<<*interface*>> *(p. 175)* **DDS::Listener** (p. 952) for status about a subscriber) 1226
DDS::SubscriberQos (QoS policies supported by a **DDS::Subscriber** (p. 1201) entity) 1230
DDS::SubscriberSeq (Declares IDL `sequence` < **DDS::Subscriber** (p. 1201) >) 1232
DDS::SubscriptionBuiltinTopicData (Entry created when a **DDS::DataReader** (p. 433) is discovered in association with its **Subscriber** (p. 1201)) 1233
DDS::SubscriptionBuiltinTopicDataDataReader (Instantiates `DataReader` (p. 433) < **DDS::SubscriptionBuiltinTopicData** (p. 1233) >) 1241
DDS::SubscriptionBuiltinTopicDataSeq (Instantiates `DDS::Sequence` (p. 1163) < **DDS::SubscriptionBuiltinTopicData** (p. 1233) >) 1242
DDS::SubscriptionBuiltinTopicDataTypeSupport (Instantiates `TypeSupport` (p. 1385) < **DDS::SubscriptionBuiltinTopicData** (p. 1233) >) 1243
DDS::SubscriptionMatchedStatus (DDS::StatusKind::SUBSCRIPTION_MATCHED_STATUS) 1244
DDS::SystemResourceLimitsQosPolicy (Configures **DDS::DomainParticipant** (p. 577)-independent resources used by RTI Data Distribution Service. Mainly used to change the maximum number of **DDS::DomainParticipant** (p. 577) entities that can be created within a single process (address space)) 1247
DDS::ThreadSettings_t (The properties of a thread of execution) . 1249
DDS::Time_t (Type for *time* representation) 1252
DDS::TimeBasedFilterQosPolicy (Filter that allows a **DDS::DataReader** (p. 433) to specify that it is interested only in (potentially) a subset of the values of the data) 1254
DDS::Topic (<<*interface*>> *(p. 175)* The most basic description of the data to be published and subscribed) 1258
DDS::TopicBuiltinTopicData (Entry created when a **Topic** (p. 1258) object discovered) 1268
DDS::TopicBuiltinTopicDataDataReader (Instantiates `DataReader` (p. 433) < **DDS::TopicBuiltinTopicData** (p. 1268) >) 1273
DDS::TopicBuiltinTopicDataSeq (Instantiates `DDS::Sequence` (p. 1163) < **DDS::TopicBuiltinTopicData** (p. 1268) >) . 1274
DDS::TopicBuiltinTopicDataTypeSupport (Instantiates `TypeSupport` (p. 1385) < **DDS::TopicBuiltinTopicData** (p. 1268) >) 1275

DDS::TopicDataQosPolicy (Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 42) during discovery) 1276
DDS::TopicListener (<<*interface*>> *(p. 175)* **DDS::Listener** (p. 952) for **DDS::Topic** (p. 1258) entities) 1278
DDS::TopicQos (QoS policies supported by a **DDS::Topic** (p. 1258) entity) 1280
DDS::TransportBuiltinKindAlias (Bits in DDS::TransportBuiltinKindMask) 1284
DDS::TransportBuiltinQosPolicy (Specifies which built-in transports are used) 1285
DDS::TransportMulticastQosPolicy (Specifies the multicast address on which a **DDS::DataReader** (p. 433) 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. 577) level) transports with which to receive the multicast data) 1287
DDS::TransportMulticastSettings_t (Type representing a list of multicast locators) 1289
DDS::TransportMulticastSettingsSeq (Declares IDL `sequence<` **DDS::TransportMulticastSettings_t** (p. 1289) >) 1291
DDS::TransportPriorityQosPolicy (This QoS policy allows the application to take advantage of transports that are capable of sending messages with different priorities) 1292
DDS::TransportSelectionQosPolicy (Specifies the physical transports a **DDS::DataWriter** (p. 499) or **DDS::DataReader** (p. 433) may use to send or receive data) 1294
DDS::TransportUnicastQosPolicy (Specifies a subset of transports and a port number that can be used by an **Entity** (p. 845) to receive data) 1296
DDS::TransportUnicastSettings_t (Type representing a list of unicast locators) 1298
DDS::TransportUnicastSettingsSeq (Declares IDL `sequence<` **DDS::TransportUnicastSettings_t** (p. 1298) >) 1300
DDS::TypeCode (The definition of a particular data type, which you can use to inspect the name, members, and other properties of types generated with **rtiddsgen** (p. 196) or to modify types you define yourself at runtime) 1301
DDS::TypeCodeFactory (A singleton factory for creating, copying, and deleting data type definitions dynamically) 1327
DDS::TypedDataReader< **T** > (<<*interface*>> *(p. 175)* <<*generic*>> *(p. 175)* User data type-specific data reader) 1338
DDS::TypedDataWriter< **T** > (<<*interface*>> *(p. 175)* <<*generic*>> *(p. 175)* User data type specific data writer) 1368

DDS::TypeSupport (<<*interface*>> *(p. 175)* An abstract *marker* interface that has to be specialized for each concrete user data type that will be used by the application) 1385
DDS::TypeSupportQosPolicy (Allows you to attach application-specific values to a **DataWriter** (p. 499) or **DataReader** (p. 433) that are passed to the serialization or deserialization routine of the associated data type) 1386
DDS::UDPv4Transport (Built-in transport plug-in using UDP/IPv4) 1388
DDS::UDPv6Transport (Built-in transport plug-in using UDP/IPv6) 1391
DDS::UnionMember (A description of a member of a union) . . . 1394
DDS::UnionMemberSeq (Defines a sequence of union members) . 1396
DDS::UnsignedIntSeq (Instantiates `DDS::Sequence` (p. 1163) < System::UInt32 >) 1397
DDS::UnsignedLongSeq (Instantiates `DDS::Sequence` (p. 1163) < System::UInt64 >) 1399
DDS::UnsignedShortSeq (Instantiates `DDS::Sequence` (p. 1163) < System::UInt16 >) 1401
DDS::UserDataQosPolicy (Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 42) during discovery) 1403
DDS::ValueMember (A description of a member of a value type) . 1405
DDS::ValueMemberSeq (Defines a sequence of value members) . . 1407
DDS::VendorId_t (<<*eXtension*>> *(p. 174)* Type used to represent the vendor of the service implementing the RTPS protocol)1408
DDS::ViewStateKind (Indicates whether or not an instance is new) 1409
DDS::WaitSet (<<*interface*>> *(p. 175)* Allows an application to wait until one or more of the attached **DDS::Condition** (p. 408) objects has a `trigger_value` of true or else until the timeout expires) 1411
DDS::WaitSetProperty_t (<<*eXtension*>> *(p. 174)* Specifies the **DDS::WaitSet** (p. 1411) behavior for multiple trigger events) 1419
DDS::WcharSeq (Instantiates `DDS::Sequence` (p. 1163) < System::Char >) 1421
DDS::WireProtocolQosPolicy (Specifies the wire-protocol-related attributes for the **DDS::DomainParticipant** (p. 577)) . . . 1423
DDS::WriterDataLifecycleQosPolicy (Controls how a **DDS::DataWriter** (p. 499) handles the lifecycle of the instances (keys) that it is registered to manage) 1431
DDS::WstringSeq (Instantiates `DDS::Sequence` (p. 1163) < System::Char* >) 1434

Chapter 5

Module Documentation

5.1 Clock Selection

APIs related to clock selection. RTI Data Distribution Service 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. 637).

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 realtime 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. 357) QoS policy associated with the **DDS::DomainParticipant** (p. 577).

See also:

DDS::PropertyQosPolicy (p. 1023)

The following table lists the supported clock selection properties.

Property	Description
dds.clock.external_clock	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"
dds.clock.internal_clock	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"

Table 5.1: *Clock Selection Properties*

5.2 Domain Module

Contains the **DDS::DomainParticipant** (p. 577) class that acts as an entry-point of RTI Data Distribution Service and acts as a factory for many of the classes. The **DDS::DomainParticipant** (p. 577) also acts as a container for the other objects that make up RTI Data Distribution Service.

Modules

- **DomainParticipantFactory**

  DDS::DomainParticipantFactory *(p. 649) entity and associated elements*

- **DomainParticipants**

  DDS::DomainParticipant *(p. 577) entity and associated elements*

- **Built-in Topics**

  Built-in objects created by RTI Data Distribution Service but accessible to the application.

5.2.1 Detailed Description

Contains the **DDS::DomainParticipant** (p. 577) class that acts as an entry-point of RTI Data Distribution Service and acts as a factory for many of the classes. The **DDS::DomainParticipant** (p. 577) also acts as a container for the other objects that make up RTI Data Distribution Service.

5.3 DomainParticipantFactory

DDS::DomainParticipantFactory (p. 649) entity and associated elements

Classes

- ^ class **DDS::DomainParticipantFactoryQos**

  *QoS policies supported by a **DDS::DomainParticipantFactory*** (p. 649).

- ^ class **DDS::DomainParticipantFactory**

  <<**singleton**>> (p. 175) <<**interface**>> (p. 175) *Allows creation and destruction of **DDS::DomainParticipant*** (p. 577) *objects.*

Properties

- ^ static DomainParticipantQos^ **DDS::DomainParticipantFactory::PARTICIPANT_QOS_DEFAULT** `[get]`

  *Special value for creating a **DomainParticipant*** (p. 577) *with default QoS.*

5.3.1 Detailed Description

DDS::DomainParticipantFactory (p. 649) entity and associated elements

5.3.2 Properties

5.3.2.1 DomainParticipantQos^ DDS::DomainParticipantFactory::PARTICIPANT_QOS_DEFAULT `[static, get, inherited]`

Special value for creating a **DomainParticipant** (p. 577) with default QoS.

When used in **DDS::DomainParticipantFactory::create_participant** (p. 665), this special value is used to indicate that the **DDS::DomainParticipant** (p. 577) should be created with the default **DDS::DomainParticipant** (p. 577) QoS by means of the operation **DDS::DomainParticipantFactory::get_default_participant_qos()** (p. 656) and using the resulting QoS to create the **DDS::DomainParticipant** (p. 577).

When used in **DDS::DomainParticipantFactory::set_default_participant_qos** (p. 654), 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. 654) operation had never been called.

When used in **DDS::DomainParticipant::set_qos** (p. 642), this special value is used to indicate that the QoS of the **DDS::DomainParticipant** (p. 577) should be changed to match the current default QoS set in the **DDS::DomainParticipantFactory** (p. 649) that the **DDS::DomainParticipant** (p. 577) belongs to.

RTI Data Distribution Service treats this special value as a constant.

Note: You cannot use this value to get the default QoS values from the domain participant factory; for this purpose, use **DDS::DomainParticipantFactory::get_default_participant_qos** (p. 656).

See also:

NDDS_DISCOVERY_PEERS (p. 312)
DDS::DomainParticipantFactory::create_participant() (p. 665)
DDS::DomainParticipantFactory::set_default_participant_qos() (p. 654)
DDS::DomainParticipant::set_qos() (p. 642)

Examples:

HelloWorld_publisher.cpp, and **HelloWorld_subscriber.cpp**.

5.4 DomainParticipants

DDS::DomainParticipant (p. 577) entity and associated elements

Classes

- ^ class **DDS::DomainParticipantQos**

  *QoS policies supported by a **DDS::DomainParticipant** (p. 577) entity.*

- ^ class **DDS::DomainParticipantListener**

  <<**interface**>> (p. 175) ***Listener*** *(p. 952) for participant status.*

- ^ class **DDS::DomainParticipant**

  <<**interface**>> (p. 175) *Container for all **DDS::DomainEntity** (p. 576) objects.*

Properties

- ^ static PublisherQos^ **DDS::DomainParticipant::PUBLISHER_QOS_DEFAULT** `[get]`

  *Special value for creating a **DDS::Publisher** (p. 1044) with default QoS.*

- ^ static SubscriberQos^ **DDS::DomainParticipant::SUBSCRIBER_QOS_DEFAULT** `[get]`

  *Special value for creating a **DDS::Subscriber** (p. 1201) with default QoS.*

- ^ static TopicQos^ **DDS::DomainParticipant::TOPIC_QOS_DEFAULT** `[get]`

  *Special value for creating a **DDS::Topic** (p. 1258) with default QoS.*

- ^ static FlowControllerProperty_t^ **DDS::DomainParticipant::FLOW_CONTROLLER_PROPERTY_DEFAULT** `[get]`

  <<**eXtension**>> (p. 174) *Special value for creating a **DDS::FlowController** (p. 867) with default property.*

- ^ static System::String^ **DDS::DomainParticipant::SQLFILTER_NAME** `[get]`

  <<**eXtension**>> (p. 174) *The name of the built-in SQL filter that can be used with ContentFilteredTopics and MultiChannel DataWriters.*

- ^ static System::String^ **DDS::DomainParticipant::STRINGMATCHFILTER_NAME** `[get]`

<<**eXtension**>> (p. 174) *The name of the built-in StringMatch filter that can be used with ContentFilteredTopics and MultiChannel DataWriters.*

5.4.1 Detailed Description

DDS::DomainParticipant (p. 577) entity and associated elements

5.4.2 Properties

5.4.2.1 PublisherQos^ DDS::DomainParticipant::PUBLISHER_QOS_DEFAULT [static, get, inherited]

Special value for creating a **DDS::Publisher** (p. 1044) with default QoS.

When used in **DDS::DomainParticipant::create_publisher** (p. 615), this special value is used to indicate that the **DDS::Publisher** (p. 1044) should be created with the default **DDS::Publisher** (p. 1044) QoS by means of the operation get_default_publisher_qos and using the resulting QoS to create the **DDS::Publisher** (p. 1044).

When used in **DDS::DomainParticipant::set_default_publisher_qos** (p. 610), 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. 610) operation had never been called.

When used in **DDS::Publisher::set_qos** (p. 1063), this special value is used to indicate that the QoS of the **DDS::Publisher** (p. 1044) should be changed to match the current default QoS set in the **DDS::DomainParticipant** (p. 577) that the **DDS::Publisher** (p. 1044) belongs to.

See also:

DDS::DomainParticipant::create_publisher (p. 615)
DDS::DomainParticipant::set_default_publisher_qos (p. 610)
DDS::Publisher::set_qos (p. 1063)

Examples:

HelloWorld_publisher.cpp.

5.4.2.2 SubscriberQos^ DDS::DomainParticipant::SUBSCRIBER_QOS_DEFAULT [static, get, inherited]

Special value for creating a **DDS::Subscriber** (p. 1201) with default QoS.

When used in **DDS::DomainParticipant::create_subscriber** (p. 618), this special value is used to indicate that the **DDS::Subscriber** (p. 1201) should be created with the default **DDS::Subscriber** (p. 1201) QoS by means of the operation get_default_subscriber_qos and using the resulting QoS to create the **DDS::Subscriber** (p. 1201).

When used in **DDS::DomainParticipant::set_default_subscriber_qos** (p. 613), 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. 613) operation had never been called.

When used in **DDS::Subscriber::set_qos** (p. 1221), this special value is used to indicate that the QoS of the **DDS::Subscriber** (p. 1201) should be changed to match the current default QoS set in the **DDS::DomainParticipant** (p. 577) that the **DDS::Subscriber** (p. 1201) belongs to.

See also:

DDS::DomainParticipant::create_subscriber (p. 618)
DDS::DomainParticipant::get_default_subscriber_qos (p. 612)
DDS::Subscriber::set_qos (p. 1221)

Examples:

HelloWorld_subscriber.cpp.

5.4.2.3 TopicQos^ DDS::DomainParticipant::TOPIC_QOS_DEFAULT [static, get, inherited]

Special value for creating a **DDS::Topic** (p. 1258) with default QoS.

When used in **DDS::DomainParticipant::create_topic** (p. 621), this special value is used to indicate that the **DDS::Topic** (p. 1258) should be created with the default **DDS::Topic** (p. 1258) QoS by means of the operation get_default_topic_qos and using the resulting QoS to create the **DDS::Topic** (p. 1258).

When used in **DDS::DomainParticipant::set_default_topic_qos** (p. 608), 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. 608) operation had never been called.

When used in **DDS::Topic::set_qos** (p. 1261), this special value is used to indicate that the QoS of the **DDS::Topic** (p. 1258) should be changed to match the current default QoS set in the **DDS::DomainParticipant** (p. 577) that the **DDS::Topic** (p. 1258) belongs to.

See also:

DDS::DomainParticipant::create_topic (p. 621)
DDS::DomainParticipant::set_default_topic_qos (p. 608)
DDS::Topic::set_qos (p. 1261)

Examples:

HelloWorld_publisher.cpp, and **HelloWorld_subscriber.cpp**.

5.4.2.4 FlowControllerProperty_t^ DDS::DomainParticipant::FLOW_-CONTROLLER_PROPERTY_DEFAULT [static, get, inherited]

<<*eXtension*>> *(p. 174)* Special value for creating a **DDS::FlowController** (p. 867) with default property.

When used in **DDS::DomainParticipant::create_flowcontroller** (p. 630), this special value is used to indicate that the **DDS::FlowController** (p. 867) should be created with the default **DDS::FlowController** (p. 867) property by means of the operation get_default_flowcontroller_property and using the resulting QoS to create the **DDS::FlowControllerProperty_t** (p. 871).

When used in **DDS::DomainParticipant::set_default_flowcontroller_-property** (p. 592), 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_flowcontroller_property** (p. 592) operation had never been called.

When used in **DDS::FlowController::set_property** (p. 868), this special value is used to indicate that the property of the **DDS::FlowController** (p. 867) should be changed to match the current default property set in the **DDS::DomainParticipant** (p. 577) that the **DDS::FlowController** (p. 867) belongs to.

See also:

DDS::DomainParticipant::create_flowcontroller (p. 630)
DDS::DomainParticipant::set_default_flowcontroller_property (p. 592)
DDS::FlowController::set_property (p. 868)

5.4.2.5 System:: String^ DDS::DomainParticipant::SQLFILTER_-NAME [static, get, inherited]

<<*eXtension*>> *(p. 174)* The name of the built-in SQL filter that can be

used with ContentFilteredTopics and MultiChannel DataWriters.

See also:

Queries and Filters Syntax (p. 184)

5.4.2.6 System:: String^ DDS::DomainParticipant::STRINGMATCHFILTER_NAME [static, get, inherited]

<<*eXtension*>> *(p. 174)* The name of the built-in StringMatch filter that can be used with ContentFilteredTopics and MultiChannel DataWriters.

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

See also:

Queries and Filters Syntax (p. 184)

5.5 Built-in Topics

Built-in objects created by RTI Data Distribution Service but accessible to the application.

Modules

- **Participant Built-in Topics**

  Builtin topic for accessing information about the DomainParticipants discovered by RTI Data Distribution Service.

- **Topic Built-in Topics**

  Builtin topic for accessing information about the Topics discovered by RTI Data Distribution Service.

- **Publication Built-in Topics**

  Builtin topic for accessing information about the Publications discovered by RTI Data Distribution Service.

- **Subscription Built-in Topics**

  Builtin topic for accessing information about the Subscriptions discovered by RTI Data Distribution Service.

Classes

- class **DDS::Locator_t**

  <<**eXtension**>> (p. 174) *Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports.*

- class **DDS::LocatorSeq**

  Declares IDL `sequence` < ***DDS::Locator_t*** *(p. 968)* >.

- struct **DDS::ProtocolVersion_t**

  <<**eXtension**>> (p. 174) *Type used to represent the version of the RTPS protocol.*

- struct **DDS::VendorId_t**

  <<**eXtension**>> (p. 174) *Type used to represent the vendor of the service implementing the RTPS protocol.*

- struct **DDS::ProductVersion_t**

<<**eXtension**>> (p. 174) *Type used to represent the current version of RTI Data Distribution Service.*

^ struct **DDS::BuiltinTopicKey_t**

The key type of the built-in topic types.

^ class **DDS::ContentFilterProperty_t**

<<**eXtension**>> (p. 174) *Type used to provide all the required information to enable content filtering.*

Properties

^ static System::Int32 **DDS::Locator_t::LOCATOR_ADDRESS_LENGTH_MAX** [get]

Declares length of address field in locator.

^ static **Locator_t**^ **DDS::Locator_t::LOCATOR_INVALID** [get]

An invalid locator.

^ static System::Int32 **DDS::Locator_t::LOCATOR_KIND_INVALID** [get]

Locator of this kind is invalid.

^ static System::UInt32 **DDS::Locator_t::LOCATOR_PORT_INVALID** [get]

An invalid port.

^ static array< System::Byte >^ **DDS::Locator_t::LOCATOR_ADDRESS_INVALID** [get]

An invalid address.

^ static System::Int32 **DDS::Locator_t::LOCATOR_KIND_UDPv4** [get]

A locator for a UDPv4 address.

^ static System::Int32 **DDS::Locator_t::LOCATOR_KIND_UDPv6** [get]

A locator for a UDPv6 address.

^ static System::Int32 **DDS::Locator_t::LOCATOR_KIND_RESERVED** [get]

Locator of this kind is reserved.

^ static System::Int32 **DDS::Locator_t::LOCATOR_KIND_SHMEM** [get]

A locator for an address acessed via shared memory.

^ static **ProtocolVersion_t DDS::ProtocolVersion_t::PROTOCOLVERSION** [get]

The most recent protocol version. Currently 1.2.

^ static **ProtocolVersion_t DDS::ProtocolVersion_t::PROTOCOLVERSION_1_0** [get]

The protocol version 1.0.

^ static **ProtocolVersion_t DDS::ProtocolVersion_t::PROTOCOLVERSION_1_1** [get]

The protocol version 1.1.

^ static **ProtocolVersion_t DDS::ProtocolVersion_t::PROTOCOLVERSION_1_2** [get]

The protocol version 1.2.

^ static **ProtocolVersion_t DDS::ProtocolVersion_t::PROTOCOLVERSION_2_0** [get]

The protocol version 2.0.

^ static **ProtocolVersion_t DDS::ProtocolVersion_t::PROTOCOLVERSION_2_1** [get]

The protocol version 2.1.

^ static **VendorId_t DDS::VendorId_t::VENDORID_UNKNOWN** [get]

The ID used when the vendor of the service implementing the RTPS protocol is not known.

^ static System::Int32 **DDS::VendorId_t::VENDORID_LENGTH_MAX** [get]

Length of vendor id.

^ static **ProductVersion_t DDS::ProductVersion_t::PRODUCTVERSION_UNKNOWN** [get]

The value used when the product version is unknown.

5.5.1 Detailed Description

Built-in objects created by RTI Data Distribution Service but accessible to the application.

RTI Data Distribution Service 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. 433) objects are introduced to be used by the application to access these discovery information.

The information can be accessed as if it was normal application data. This allows the application to know when there are any changes in those values by means of the **DDS::Listener** (p. 952) or the **DDS::Condition** (p. 408) mechanisms.

The built-in data-readers all belong to a built-in **DDS::Subscriber** (p. 1201), which can be retrieved by using the method **DDS::DomainParticipant::get_builtin_subscriber** (p. 632). The built-in **DDS::DataReader** (p. 433) objects can be retrieved by using the operation **DDS::Subscriber::lookup_datareader** (p. 1215), with the topic name as a parameter.

Built-in entities have default listener settings as well. The built-in **DDS::Subscriber** (p. 1201) 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. This 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. 433) will not provide data pertaining to entities created from the same **DDS::DomainParticipant** (p. 577) under the assumption that such entities are already known to the application that created them.

Refer to **DDS::ParticipantBuiltinTopicData** (p. 1002), **DDS::TopicBuiltinTopicData** (p. 1268), **DDS::SubscriptionBuiltinTopicData** (p. 1233) and **DDS::PublicationBuiltinTopicData** (p. 1030) for a description of all the built-in topics and their contents.

The QoS of the built-in **DDS::Subscriber** (p. 1201) and **DDS::DataReader** (p. 433) objects is given by the following table:

5.5.2 Properties

5.5.2.1 System:: Int32 DDS::Locator_t::LOCATOR_ADDRESS_LENGTH_MAX [static, get, inherited]

Declares length of address field in locator.

5.5.2.2 Locator_t^ DDS::Locator_t::LOCATOR_INVALID [static, get, inherited]

An invalid locator.

5.5.2.3 System:: Int32 DDS::Locator_t::LOCATOR_KIND_INVALID [static, get, inherited]

Locator of this kind is invalid.

5.5.2.4 System:: UInt32 DDS::Locator_t::LOCATOR_PORT_INVALID [static, get, inherited]

An invalid port.

5.5.2.5 array< System:: Byte>^ DDS::Locator_t::LOCATOR_ADDRESS_INVALID [static, get, inherited]

An invalid address.

5.5.2.6 System:: Int32 DDS::Locator_t::LOCATOR_KIND_UDPv4 [static, get, inherited]

A locator for a UDPv4 address.

5.5.2.7 System:: Int32 DDS::Locator_t::LOCATOR_KIND_UDPv6 [static, get, inherited]

A locator for a UDPv6 address.

5.5.2.8 System:: Int32 DDS::Locator_t::LOCATOR_KIND_RESERVED [static, get, inherited]

Locator of this kind is reserved.

5.5.2.9 System:: Int32 DDS::Locator_t::LOCATOR_KIND_SHMEM [static, get, inherited]

A locator for an address acessed via shared memory.

5.5.2.10 ProtocolVersion_t DDS::ProtocolVersion_t::PROTOCOLVERSION [static, get, inherited]

The most recent protocol version. Currently 1.2.

5.5.2.11 ProtocolVersion_t DDS::ProtocolVersion_t::PROTOCOLVERSION_1_0 [static, get, inherited]

The protocol version 1.0.

5.5.2.12 ProtocolVersion_t DDS::ProtocolVersion_t::PROTOCOLVERSION_1_1 [static, get, inherited]

The protocol version 1.1.

5.5.2.13 ProtocolVersion_t DDS::ProtocolVersion_t::PROTOCOLVERSION_1_2 [static, get, inherited]

The protocol version 1.2.

5.5.2.14 ProtocolVersion_t DDS::ProtocolVersion_t::PROTOCOLVERSION_2_0 [static, get, inherited]

The protocol version 2.0.

5.5.2.15 ProtocolVersion_t DDS::ProtocolVersion_-t::PROTOCOLVERSION_2_1 [static, get, inherited]

The protocol version 2.1.

5.5.2.16 VendorId_t DDS::VendorId_t::VENDORID_UNKNOWN [static, get, inherited]

The ID used when the vendor of the service implementing the RTPS protocol is not known.

5.5.2.17 System:: Int32 DDS::VendorId_t::VENDORID_-LENGTH_MAX [static, get, inherited]

Length of vendor id.

5.5.2.18 ProductVersion_t DDS::ProductVersion_-t::PRODUCTVERSION_UNKNOWN [static, get, inherited]

The value used when the product version is unknown.

QoS	Value
DDS::UserDataQosPolicy (p. 1403)	0-length sequence
DDS::TopicDataQosPolicy (p. 1276)	0-length sequence
DDS::GroupDataQosPolicy (p. 890)	0-length sequence
DDS::DurabilityQosPolicy (p. 709)	DDS::DurabilityQosPolicyKind::TRANSIENT_-LOCAL_DURABILITY_QOS
DDS::DurabilityServiceQosPolicy (p. 714)	Does not apply as DDS::DurabilityQosPolicyKind is DDS::DurabilityQosPolicyKind::TRANSIENT_-LOCAL_DURABILITY_QOS
DDS::PresentationQosPolicy (p. 1012)	access_scope = DDS::PresentationQosPolicyAccessScopeKind::TOPIC_-PRESENTATION_QOS coherent_access = false ordered_access = false
DDS::DeadlineQosPolicy (p. 557)	Period = infinite
DDS::LatencyBudgetQosPolicy (p. 948)	duration = 0
DDS::OwnershipQosPolicy (p. 993)	DDS::OwnershipQosPolicyKind::SHARED_-OWNERSHIP_QOS
DDS::OwnershipStrengthQosPolicy (p. 1000)	value = 0
DDS::LivelinessQosPolicy (p. 960)	kind = DDS::LivelinessQosPolicyKind::AUTOMATIC_-LIVELINESS_QOS lease_duration = 0
DDS::TimeBasedFilterQosPolicy (p. 1254)	minimum_separation = 0
DDS::PartitionQosPolicy (p. 1008)	0-length sequence
DDS::ReliabilityQosPolicy (p. 1094)	kind = DDS::ReliabilityQosPolicyKind::RELIABLE_-RELIABILITY_QOS max_blocking_time = 100 milliseconds
DDS::DestinationOrderQosPolicy (p. 560)	DDS::DestinationOrderQosPolicyKind::BY_-RECEPTION_TIMESTAMP_-DESTINATIONORDER_QOS
DDS::HistoryQosPolicy (p. 898)	kind = DDS::HistoryQosPolicyKind::KEEP_-LAST_HISTORY_QOS depth = 1
DDS::ResourceLimitsQosPolicy (p. 1109)	max_samples = DDS::LENGTH_UNLIMITED max_instances = DDS::LENGTH_UNLIMITED

5.6 Topic Module

Contains the **DDS::Topic** (p. 1258), **DDS::ContentFilteredTopic** (p. 419), and **DDS::MultiTopic** (p. 984) classes, the **DDS::TopicListener** (p. 1278) interface, and more generally, all that is needed by an application to define **DDS::Topic** (p. 1258) objects and attach QoS policies to them.

Modules

- **Topics**

  DDS::Topic *(p. 1258) entity and associated elements*

- **User Data Type Support**

  Defines generic classes and macros to support user data types.

- **Type Code Support**

  <<**eXtension**>> (p. 174) *A* **DDS::TypeCode** (p. 1301) *is a mechanism for representing a type at runtime. RTI Data Distribution Service 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. 73) capability or to inspect the type information you receive from remote readers and writers.*

- **Built-in Types**

  <<**eXtension**>> (p. 174) *RTI Data Distribution Service provides a set of very simple data types for you to use with the topics in your application.*

- **Dynamic Data**

  <<**eXtension**>> (p. 174) *The Dynamic Data API provides a way to interact with arbitrarily complex data types at runtime without the need for code generation.*

5.6.1 Detailed Description

Contains the **DDS::Topic** (p. 1258), **DDS::ContentFilteredTopic** (p. 419), and **DDS::MultiTopic** (p. 984) classes, the **DDS::TopicListener** (p. 1278) interface, and more generally, all that is needed by an application to define **DDS::Topic** (p. 1258) objects and attach QoS policies to them.

5.7 Topics

DDS::Topic (p. 1258) entity and associated elements

Classes

- struct **DDS::InconsistentTopicStatus**

  DDS::StatusKind::INCONSISTENT_TOPIC_STATUS

- class **DDS::TopicQos**

  *QoS policies supported by a **DDS::Topic** (p. 1258) entity.*

- interface **DDS::ITopicDescription**

  <<**interface**>> (p. 175) *Base class for **DDS::Topic** (p. 1258), **DDS::ContentFilteredTopic** (p. 419), and **DDS::MultiTopic** (p. 984).*

- class **DDS::ContentFilteredTopic**

  <<**interface**>> (p. 175) *Specialization of DDS::TopicDescription that allows for content-based subscriptions.*

- class **DDS::MultiTopic**

  *[**Not supported (optional)**]* <<**interface**>> (p. 175) *A specialization of DDS::TopicDescription that allows subscriptions that combine/filter/rearrange data coming from several topics.*

- class **DDS::Topic**

  <<**interface**>> (p. 175) *The most basic description of the data to be published and subscribed.*

- class **DDS::TopicListener**

  <<**interface**>> (p. 175) ***DDS::Listener*** (p. 952) *for **DDS::Topic*** (p. 1258) *entities.*

5.7.1 Detailed Description

DDS::Topic (p. 1258) entity and associated elements

5.8 User Data Type Support

Defines generic classes and macros to support user data types.

Classes

- struct **DDS::InstanceHandle_t**

  Type definition for an instance handle.

- class **DDS::InstanceHandleSeq**

  Instantiates `DDS::Sequence` (p. 1163) < ***DDS::InstanceHandle_t*** (p. 905) > .

- class **DDS::TypeSupport**

  <<**interface**>> (p. 175) *An abstract* marker *interface that has to be specialized for each concrete user data type that will be used by the application.*

- struct **Foo**

  A representative user-defined data type.

- class **FooTypeSupport**

  <<**interface**>> (p. 175) <<**generic**>> (p. 175) *User data type specific interface.*

Variables

- static **InstanceHandle_t DDS::InstanceHandle_t::HANDLE_NIL**

  The NIL instance handle.

Properties

- bool **DDS::InstanceHandle_t::is_nil** `[get]`

  Compare this handle to ***DDS::InstanceHandle_t::HANDLE_NIL*** (p. 53).

5.8.1 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 Data Distribution Service.

RTI Data Distribution Service provides an automatic means to generate all these type-specific classes with the **rtiddsgen** (p. 196) utility. The complete set of automatic classes created for a hypothetical user data type named `Foo` (p. 877) are shown below.

The macros defined here declare the strongly typed APIs needed to support an arbitrary user defined data of type `Foo` (p. 877).

See also:

rtiddsgen (p. 196)

5.8.2 Variable Documentation

5.8.2.1 InstanceHandle_t DDS::InstanceHandle_t::HANDLE_NIL [static, inherited]

The NIL instance handle.

Special **DDS::InstanceHandle_t** (p. 905) value

See also:

DDS::InstanceHandle_t::is_nil (p. 53)

Examples:

HelloWorld_publisher.cpp.

5.8.3 Properties

5.8.3.1 bool DDS::InstanceHandle_t::is_nil [get, inherited]

Compare this handle to **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53).

Returns:

true if the given instance handle is equal to **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) or false otherwise.

See also:

DDS::InstanceHandle::Equals

5.9 Type Code Support

*<<**eXtension**>> (p. 174)* A ***DDS::TypeCode*** *(p. 1301)* is a mechanism for representing a type at runtime. RTI Data Distribution Service 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. 73) capability or to inspect the type information you receive from remote readers and writers.

Classes

- class **DDS::TypeCode**

  *The definition of a particular data type, which you can use to inspect the name, members, and other properties of types generated with **rtiddsgen** (p. 196) or to modify types you define yourself at runtime.*

- class **DDS::StructMember**

  A description of a member of a struct.

- class **DDS::StructMemberSeq**

  Defines a sequence of struct members.

- class **DDS::UnionMember**

  A description of a member of a union.

- class **DDS::UnionMemberSeq**

  Defines a sequence of union members.

- class **DDS::EnumMember**

  A description of a member of an enumeration.

- class **DDS::EnumMemberSeq**

  Defines a sequence of enumerator members.

- class **DDS::ValueMember**

  A description of a member of a value type.

- class **DDS::ValueMemberSeq**

  Defines a sequence of value members.

- class **DDS::TypeCodeFactory**

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

Enumerations

^ 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_SPARSE }

Enumeration type for ***DDS::TypeCode*** *(p. 1301) kinds.*

^ enum **DDS::ValueModifier** {
DDS::VM_NONE,
DDS::VM_CUSTOM,
DDS::VM_ABSTRACT,
DDS::VM_TRUNCATABLE }

Modifier type for a value type.

^ enum **DDS::Visibility** {
DDS::PRIVATE_MEMBER,
DDS::PUBLIC_MEMBER }

Type to indicate the visibility of a value type member.

Variables

^ TypeCode^ **DDS::TypeCode::TC_NULL**

Basic null type.

^ TypeCode^ **DDS::TypeCode::TC_LONG**

Basic 32-bit signed integer type.

^ TypeCode^ **DDS::TypeCode::TC_USHORT**

Basic unsigned 16-bit integer type.

^ TypeCode^ **DDS::TypeCode::TC_ULONG**

Basic unsigned 32-bit integer type.

^ TypeCode^ **DDS::TypeCode::TC_FLOAT**

Basic 32-bit floating point type.

^ TypeCode^ **DDS::TypeCode::TC_DOUBLE**

Basic 64-bit floating point type.

^ TypeCode^ **DDS::TypeCode::TC_BOOLEAN**

Basic Boolean type.

^ TypeCode^ **DDS::TypeCode::TC_OCTET**

Basic octet/byte type.

^ TypeCode^ **DDS::TypeCode::TC_LONGLONG**

Basic 64-bit integer type.

^ TypeCode^ **DDS::TypeCode::TC_ULONGLONG**

Basic unsigned 64-bit integer type.

^ TypeCode^ **DDS::TypeCode::TC_LONGDOUBLE**

Basic 128-bit floating point type.

^ TypeCode^ **DDS::TypeCode::TC_WCHAR**

Basic four-byte character type.

- System::Int32 **DDS::TypeCode::MEMBER_ID_INVALID**

  *A sentinel indicating an invalid **DDS::TypeCode** (p. 1301) member ID.*

- System::UInt32 **DDS::TypeCode::INDEX_INVALID**

  *A sentinel indicating an invalid **DDS::TypeCode** (p. 1301) member index.*

- System::Int16 **DDS::TypeCode::NOT_BITFIELD**

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

- System::Byte **DDS::TypeCode::NONKEY_MEMBER**

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

- System::Byte **DDS::TypeCode::KEY_MEMBER**

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

- System::Byte **DDS::TypeCode::NONKEY_REQUIRED_MEMBER**

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

5.9.1 Detailed Description

*<<**eXtension**>> (p. 174)* A ***DDS::TypeCode*** *(p. 1301)* is a mechanism for representing a type at runtime. RTI Data Distribution Service 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. 73) 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. 1301) class, which has an analogue in CORBA.

A **DDS::TypeCode** (p. 1301) value consists of a type code *kind* (represented by the DDS::TCKind enumeration) and a list of *members* (that is, fields). These members are recursive: each one has its own **DDS::TypeCode** (p. 1301), 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::Equals method is a more reliable test than comparing the string type names, requiring equivalent definitions of the types.

5.9.2 Accessing a Local ::DDS::TypeCode

When generating types with **rtiddsgen** (p. 196), type codes are enabled by default. (The *-notypecode* option can be used to disable generation of **DDS::TypeCode** (p. 1301) information.) For these types, a **DDS::TypeCode** (p. 1301) may be accessed via the **FooTypeCode.VALUE** member.

This API also includes support for dynamic creation of **DDS::TypeCode** (p. 1301) values, typically for use with the **Dynamic Data** (p. 73) API. You can create a **DDS::TypeCode** (p. 1301) using the **DDS::TypeCodeFactory** (p. 1327) class. You will construct the **DDS::TypeCode** (p. 1301) 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. 1331), which provides the **DDS::TypeCode** (p. 1301) instances corresponding to the primitive types (e.g. DDS::TCKind::TK_LONG, DDS::TCKind::TK_SHORT, and so on).
- **DDS::TypeCodeFactory::create_string_tc** (p. 1334) and **DDS::TypeCodeFactory::create_wstring_tc** (p. 1334) create a **DDS::TypeCode** (p. 1301) representing a text string with a certain *bound* (i.e. maximum length).
- **DDS::TypeCodeFactory::create_array_tc** (p. 1335) and **DDS::TypeCodeFactory::create_sequence_tc** (p. 1335) create a **DDS::TypeCode** (p. 1301) for a collection based on the **DDS::TypeCode** (p. 1301) for its elements.
- **DDS::TypeCodeFactory::create_struct_tc** (p. 1331), **DDS::TypeCodeFactory::create_value_tc** (p. 1332), and **DDS::TypeCodeFactory::create_sparse_tc** (p. 1336) create a **DDS::TypeCode** (p. 1301) for a structured type.

5.9.3 Accessing a Remote ::DDS::TypeCode

In addition to being used locally, RTI Data Distribution Service can transmit **DDS::TypeCode** (p. 1301) 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. 73)). This functionality is useful for a generic system monitoring tool like `rtiddsspy`.

Remote **DDS::TypeCode** (p. 1301) 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. 42). Discovered **DDS::TypeCode** (p. 1301) values are not cached by RTI Data Distribution Service upon receipt and are therefore not available from the built-in topic data returned by **DDS::DataWriter::get_matched_subscription_data** (p. 510) or **DDS::DataReader::get_matched_publication_data** (p. 443).

The space available locally to deserialize a discovered remote **DDS::TypeCode** (p. 1301) is specified by the **DDS::DomainParticipant** (p. 577)'s **DDS::DomainParticipantResourceLimitsQosPolicy::type_code_max_serialized_length** (p. 703) 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. 1301)
Dynamic Data (p. 73)
rtiddsgen (p. 196)
DDS::SubscriptionBuiltinTopicData (p. 1233)
DDS::PublicationBuiltinTopicData (p. 1030)

5.9.4 Enumeration Type Documentation

5.9.4.1 enum DDS::TCKind

Enumeration type for **DDS::TypeCode** (p. 1301) kinds.

Type code kinds are modeled as values of this type.

Enumerator:

TK_NULL Indicates that a type code does not describe anything.

TK_SHORT short type.

TK_LONG long type.

TK_USHORT unsigned short type.

TK_ULONG unsigned long type.

TK_FLOAT float type.

TK_DOUBLE double type.

TK_BOOLEAN boolean type.

TK_CHAR char type.

TK_OCTET octet type.

TK_STRUCT struct type.

TK_UNION union type.

TK_ENUM enumerated type.

TK_STRING string type.

TK_SEQUENCE sequence type.

TK_ARRAY array type.

TK_ALIAS alias (typedef) type.

TK_LONGLONG long long type.

TK_ULONGLONG unsigned long long type.

TK_LONGDOUBLE long double type.

TK_WCHAR wide char type.

TK_WSTRING wide string type.

TK_VALUE value type.

TK_SPARSE A sparse value type.

A sparse value type is one in which all of the fields are not necessarily sent on the network as a part of every sample.

Fields of a sparse value type fall into one of three categories:

- Key fields (see **DDS::TypeCode::KEY_MEMBER** (p. 66))
- Non-key, but required members (see **DDS::TypeCode::NONKEY_REQUIRED_MEMBER** (p. 66))
- Non-key, optional members (see **DDS::TypeCode::NONKEY_MEMBER** (p. 65))

Fields of the first two kinds must appear in every sample. These are also the only kinds of fields on which you can perform content filtering (see **DDS::ContentFilteredTopic** (p. 419)), because filter evaluation on a non-existent field is not well defined.

5.9.4.2 enum DDS::ValueModifier

Modifier type for a value type.

See also:

DDS::ValueModifier::VM_NONE
DDS::ValueModifier::VM_CUSTOM
DDS::ValueModifier::VM_ABSTRACT
DDS::ValueModifier::VM_TRUNCATABLE

Enumerator:

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

See also:

DDS::ValueModifier

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

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

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

5.9.4.3 enum DDS::Visibility

Type to indicate the visibility of a value type member.

See also:

DDS::Visibility::PRIVATE_MEMBER
DDS::Visibility::PUBLIC_MEMBER

Enumerator:

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

See also:

DDS::Visibility
DDS::Visibility::PUBLIC_MEMBER

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

See also:

DDS::Visibility
DDS::Visibility::PRIVATE_MEMBER

5.9.5 Variable Documentation

5.9.5.1 TypeCode ^ DDS::TypeCode::TC_NULL [inherited]

Basic null type.

See also:

DDS::TypeCodeFactory::get_primitive_tc (p. 1331)

5.9.5.2 TypeCode ^ DDS::TypeCode::TC_LONG [inherited]

Basic 32-bit signed integer type.

See also:

DDS::TypeCodeFactory::get_primitive_tc (p. 1331)

5.9.5.3 TypeCode ^ DDS::TypeCode::TC_USHORT [inherited]

Basic unsigned 16-bit integer type.

See also:

DDS::TypeCodeFactory::get_primitive_tc (p. 1331)

5.9.5.4 TypeCode ^ DDS::TypeCode::TC_ULONG [inherited]

Basic unsigned 32-bit integer type.

See also:

DDS::TypeCodeFactory::get_primitive_tc (p. 1331)

5.9.5.5 TypeCode ^ DDS::TypeCode::TC_FLOAT [inherited]

Basic 32-bit floating point type.

See also:

DDS::TypeCodeFactory::get_primitive_tc (p. 1331)

5.9.5.6 TypeCode ^ DDS::TypeCode::TC_DOUBLE [inherited]

Basic 64-bit floating point type.

See also:

DDS::TypeCodeFactory::get_primitive_tc (p. 1331)

5.9.5.7 TypeCode ^ DDS::TypeCode::TC_BOOLEAN [inherited]

Basic Boolean type.

See also:

DDS::TypeCodeFactory::get_primitive_tc (p. 1331)

5.9.5.8 TypeCode ^ DDS::TypeCode::TC_OCTET [inherited]

Basic octet/byte type.

See also:

DDS::TypeCodeFactory::get_primitive_tc (p. 1331)

5.9.5.9 TypeCode ^ DDS::TypeCode::TC_LONGLONG [inherited]

Basic 64-bit integer type.

See also:

DDS::TypeCodeFactory::get_primitive_tc (p. 1331)

5.9.5.10 TypeCode ^ DDS::TypeCode::TC_ULONGLONG [inherited]

Basic unsigned 64-bit integer type.

See also:

DDS::TypeCodeFactory::get_primitive_tc (p. 1331)

5.9.5.11 TypeCode ^ DDS::TypeCode::TC_LONGDOUBLE [inherited]

Basic 128-bit floating point type.

See also:

DDS::TypeCodeFactory::get_primitive_tc (p. 1331)

5.9.5.12 TypeCode ^ DDS::TypeCode::TC_WCHAR [inherited]

Basic four-byte character type.

See also:

DDS::TypeCodeFactory::get_primitive_tc (p. 1331)

5.9.5.13 System::Int32 DDS::TypeCode::MEMBER_ID_INVALID [inherited]

A sentinel indicating an invalid **DDS::TypeCode** (p. 1301) member ID.

5.9.5.14 System::UInt32 DDS::TypeCode::INDEX_INVALID [inherited]

A sentinel indicating an invalid **DDS::TypeCode** (p. 1301) member index.

5.9.5.15 System::Int16 DDS::TypeCode::NOT_BITFIELD [inherited]

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

5.9.5.16 System::Byte DDS::TypeCode::NONKEY_MEMBER [inherited]

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

Only sparse value types (i.e. types of DDS::TCKind DDS::TCKind::TK_SPARSE) support this flag. Non-key members of other type kinds should use the flag **DDS::TypeCode::NONKEY_REQUIRED_MEMBER** (p. 66).

If a type is used with the **Dynamic Data** (p. 73) facility, a **DDS::DynamicData** (p. 719) sample of the type will only contain a

value for a **DDS::TypeCode::NONKEY_MEMBER** (p. 65) field if one has been explicitly set (see, for example, **DDS::DynamicData::set_int** (p. 776)). The middleware will *not* assume any default value.

See also:

DDS::TypeCode::KEY_MEMBER (p. 66)
DDS::TypeCode::NONKEY_REQUIRED_MEMBER (p. 66)
DDS::TypeCode::KEY_MEMBER (p. 66)
DDS::TypeCode::add_member (p. 1323)
DDS::TypeCode::add_member_ex (p. 1324)
DDS::TypeCode::is_member_key (p. 1312)
DDS::TypeCode::is_member_required (p. 1312)
DDS::StructMember::is_key (p. 1199)
DDS::ValueMember::is_key (p. 1406)

5.9.5.17 System::Byte DDS::TypeCode::KEY_MEMBER [inherited]

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. 73) facility, all **DDS::DynamicData** (p. 719) samples of the type will contain a value for all **DDS::TypeCode::KEY_MEMBER** (p. 66) fields, even if the type is a sparse value type (i.e. of kind DDS::TCKind::TK_SPARSE). If you do not set a value of the member explicitly (see, for example, **DDS::DynamicData::set_int** (p. 776)), 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. 66)
DDS::TypeCode::NONKEY_MEMBER (p. 65)
DDS::TypeCode::add_member (p. 1323)
DDS::TypeCode::add_member_ex (p. 1324)
DDS::TypeCode::is_member_key (p. 1312)
DDS::TypeCode::is_member_required (p. 1312)
DDS::StructMember::is_key (p. 1199)
DDS::ValueMember::is_key (p. 1406)

5.9.5.18 System::Byte DDS::TypeCode::NONKEY_REQUIRED_MEMBER [inherited]

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. 73) facility, all **DDS::DynamicData** (p. 719) samples of the type will contain a value for all **DDS::TypeCode::NONKEY_REQUIRED_MEMBER** (p. 66) fields, even if the type is a sparse value type (i.e. of kind DDS::TCKind::TK_SPARSE). If you do not set a value of the member explicitly (see, for example, **DDS::DynamicData::set_int** (p. 776)), 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. 66)
DDS::TypeCode::NONKEY_MEMBER (p. 65)
DDS::TypeCode::KEY_MEMBER (p. 66)
DDS::TypeCode::add_member (p. 1323)
DDS::TypeCode::add_member_ex (p. 1324)
DDS::TypeCode::is_member_key (p. 1312)
DDS::TypeCode::is_member_required (p. 1312)
DDS::StructMember::is_key (p. 1199)
DDS::ValueMember::is_key (p. 1406)

5.10 Built-in Types

<<*eXtension*>> *(p. 174)* RTI Data Distribution Service provides a set of very simple data types for you to use with the topics in your application.

Modules

- **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.

- **KeyedString Built-in Type**

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

- **String Built-in Type**

  Built-in type consisting of a single character string.

5.10.1 Detailed Description

<<*eXtension*>> *(p. 174)* RTI Data Distribution Service 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. 933): 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::Bytes** (p. 388): A payload consisting of an opaque variable-length array of bytes. This type has no key.
- **DDS::KeyedBytes** (p. 915): 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. 933) types are appropriate for simple text-based applications. The **DDS::Bytes** (p. 388) and **DDS::KeyedBytes** (p. 915) types are appropriate for applications that perform their own custom data serialization, such as legacy applications still in the process of migrating to RTI Data Distribution Service. 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. 419)).

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** (p. 196) utility
- At runtime, by using the **Dynamic Data** (p. 73) API

5.10.2 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. 1023) 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 overwrittem in the DataWriters and DataReaders.

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.

5.10.3 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.

For more information about the built-in types, including how to control memory usage and maximum lengths, please see chapter 3, *Data Types and Data Samples*, in the RTI Data Distribution Service User's Manual.

Property	Description
dds.builtin_type.string.alloc_size	Maximum size of the strings published by the **DDS::StringDataWriter** (p. 1190) or received the **DDS::StringDataReader** (p. 1186) (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. 937) or **DDS::KeyedStringDataReader** (p. 935) (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. 937) or received by the **DDS::KeyedStringDataReader** (p. 935) (includes the NULL-terminated character). Default: dds.builtin_type.keyed_string.max_size if defined. Otherwise, 1024.
dds.builtin_type.octets.alloc_size	Maximum size of the octet sequences published the **DDS::BytesDataWriter** (p. 392) or received by the **DDS::BytesDataReader** (p. 391). 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 or received by the **DDS::KeyedBytesDataReader** (p. 918) (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 or received by a **DDS::KeyedBytesDataReader** (p. 918). Default: dds.builtin_type.keyed_

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

Table 5.4: *Properties for Allocating Size of Builtin Types, per DomainParticipant*

5.11 Dynamic Data

<<***eXtension***>> *(p. 174)* The Dynamic Data API provides a way to interact with arbitrarily complex data types at runtime without the need for code generation.

Classes

- class **DDS::DynamicDataProperty_t**

  A collection of attributes used to configure ***DDS::DynamicData*** *(p. 719) objects.*

- class **DDS::DynamicDataInfo**

  A descriptor for a ***DDS::DynamicData*** *(p. 719) object.*

- class **DDS::DynamicDataMemberInfo**

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

- class **DDS::DynamicData**

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

- class **DDS::DynamicDataSeq**

  An ordered collection of ***DDS::DynamicData*** *(p. 719) elements.*

- class **DDS::DynamicDataReader**

  Reads (subscribes to) objects of type ***DDS::DynamicData*** *(p. 719).*

- class **DDS::DynamicDataWriter**

  Writes (publishes) objects of type ***DDS::DynamicData*** *(p. 719).*

- class **DDS::DynamicDataTypeSerializationProperty_t**

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

- class **DDS::DynamicDataTypeProperty_t**

  A collection of attributes used to configure ***DDS::DynamicDataTypeSupport*** *(p. 822) objects.*

- class **DDS::DynamicDataTypeSupport**

  A factory for registering a dynamically defined type and creating ***DDS::DynamicData*** *(p. 719) objects.*

Variables

- static System::Int32 **DDS::DynamicData::MEMBER_ID_UNSPECIFIED**

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

- static DynamicDataProperty_t^ **DDS::DynamicData::DYNAMIC_DATA_PROPERTY_DEFAULT**

  Sentinel constant indicating default values for ***DDS::DynamicDataProperty_t*** *(p. 813).*

- static DynamicDataTypeProperty_t^ **DDS::DynamicDataTypeProperty_t::DYNAMIC_DATA_TYPE_PROPERTY_DEFAULT**

  Sentinel constant indicating default values for ***DDS::DynamicDataTypeProperty_t*** *(p. 818).*

5.11.1 Detailed Description

*<<**eXtension**>> (p. 174)* 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. 1301) (see Type Code Support (p. 55)) that defines the type definition you want to use.

A **DDS::TypeCode** (p. 1301) includes a type's *kind* (DDS::TCKind), *name*, and *members* (that is, fields). You can create your own **DDS::TypeCode** (p. 1301) using the **DDS::TypeCodeFactory** (p. 1327) class – see, for example, the **DDS::TypeCodeFactory::create_struct_tc** (p. 1331) method. Alternatively, you can use a remote **DDS::TypeCode** (p. 1301) that you discovered on the network (see **Built-in Topics** (p. 42)) or one generated by **rtiddsgen** (p. 196).

2. Wrap the DDS::TypeCode (p. 1301) in a DDS::DynamicDataTypeSupport (p. 822) object.

See the constructor **DDS::DynamicDataTypeSupport::DynamicDataTypeSupport** (p. 823). This object lets you connect the type definition to a **DDS::DomainParticipant** (p. 577) and manage data samples (of type **DDS::DynamicData** (p. 719)).

3. Register the DDS::DynamicDataTypeSupport (p. 822) with one or more domain participants.

See **DDS::DynamicDataTypeSupport::register_type** (p. 825). 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. 1258) from the DDS::DomainParticipant (p. 577).

Use the name under which you registered your data type – see **DDS::DomainParticipant::create_topic** (p. 621). This **DDS::Topic** (p. 1258) is what you will use to produce and consume data.

5. Create a DDS::DynamicDataWriter (p. 828) and/or DDS::DynamicDataReader (p. 815).

These objects will produce and/or consume data (of type **DDS::DynamicData** (p. 719)) on the **DDS::Topic** (p. 1258). You can create these objects directly from the **DDS::DomainParticipant** (p. 577) – see **DDS::DomainParticipant::create_datawriter** (p. 599) and **DDS::DomainParticipant::create_datareader** (p. 603) – or by first creating intermediate **DDS::Publisher** (p. 1044) and **DDS::Subscriber** (p. 1201) objects – see **DDS::DomainParticipant::create_publisher** (p. 615) and **DDS::DomainParticipant::create_subscriber** (p. 618).

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. 577) is optional; all types are automatically unregistered when the **DDS::DomainParticipant** (p. 577) itself is deleted.

5.11.2 Variable Documentation

5.11.2.1 System::Int32 DDS::DynamicData::MEMBER_ID_UNSPECIFIED [static, inherited]

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.

5.11.2.2 static DynamicDataProperty_t DDS::DynamicData::DYNAMIC_DATA_PROPERTY_DEFAULT [static, get, inherited]

Sentinel constant indicating default values for **DDS::DynamicDataProperty_t** (p. 813).

Pass this object instead of your own **DDS::DynamicDataProperty_t** (p. 813) object to use the default property values:

In C#:

```
DynamicData sample = new DynamicData(
    myTypeCode,
    DynamicDataProperty_t.DYNAMIC_DATA_PROPERTY_DEFAULT);
```

In C++/CLI:

```
DynamicData^ sample = gcnew DynamicData(
    myTypeCode,
    DynamicDataProperty_t::DYNAMIC_DATA_PROPERTY_DEFAULT);
```

See also:

DDS::DynamicDataProperty_t (p. 813)

5.11.2.3 static DynamicDataTypeProperty_t DDS::DynamicDataTypeProperty_t::DYNAMIC_DATA_TYPE_PROPERTY_DEFAULT [static, get, inherited]

Sentinel constant indicating default values for **DDS::DynamicDataTypeProperty_t** (p. 818).

Pass this object instead of your own **DDS::DynamicDataTypeProperty_t** (p. 818) object to use the default property values:

In C#:

```
DynamicDataTypeSupport support = new DynamicDataTypeSupport(
    myTypeCode,
    DynamicDataTypeSupport.DYNAMIC_DATA_TYPE_PROPERTY_DEFAULT);
```

In C++/CLI:

```
DynamicDataTypeSupport^ support = gcnew DynamicDataTypeSupport(
    myTypeCode,
    DynamicDataTypeSupport::DYNAMIC_DATA_TYPE_PROPERTY_DEFAULT);
```

See also:

DDS::DynamicDataTypeProperty_t (p. 818)

5.12 Publication Module

Contains the **DDS::FlowController** (p. 867), **DDS::Publisher** (p. 1044), and **DDS::DataWriter** (p. 499) classes as well as the **DDS::PublisherListener** (p. 1069) and **DDS::DataWriterListener** (p. 524) interfaces, and more generally, all that is needed on the publication side.

Modules

- **Publishers**

  DDS::Publisher *(p. 1044) entity and associated elements*

- **Data Writers**

  DDS::DataWriter *(p. 499) entity and associated elements*

- **Flow Controllers**

  <<**eXtension**>> (p. 174) ***DDS::FlowController*** *(p. 867) and associated elements*

5.12.1 Detailed Description

Contains the **DDS::FlowController** (p. 867), **DDS::Publisher** (p. 1044), and **DDS::DataWriter** (p. 499) classes as well as the **DDS::PublisherListener** (p. 1069) and **DDS::DataWriterListener** (p. 524) interfaces, and more generally, all that is needed on the publication side.

5.13 Publishers

DDS::Publisher (p. 1044) entity and associated elements

Classes

- class **DDS::PublisherQos**

  *QoS policies supported by a **DDS::Publisher** (p. 1044) entity.*

- class **DDS::PublisherListener**

  <<**interface**>> (p. 175) ***DDS::Listener*** *(p. 952) for **DDS::Publisher** (p. 1044) status.*

- class **DDS::PublisherSeq**

  Declares IDL `sequence` *<* ***DDS::Publisher*** *(p. 1044)* *>* .

- class **DDS::Publisher**

  <<**interface**>> (p. 175) *A publisher is the object responsible for the actual dissemination of publications.*

Properties

- static DataWriterQos^ **DDS::Publisher::DATAWRITER_QOS_DEFAULT** `[get]`

  *Special value for creating **DDS::DataWriter** (p. 499) with default QoS.*

- static DataWriterQos^ **DDS::Publisher::DATAWRITER_QOS_USE_TOPIC_QOS** `[get]`

  *Special value for creating **DDS::DataWriter** (p. 499) with a combination of the default **DDS::DataWriterQos** (p. 546) and the **DDS::TopicQos** (p. 1280).*

5.13.1 Detailed Description

DDS::Publisher (p. 1044) entity and associated elements

5.13.2 Properties

5.13.2.1 DataWriterQos^ DDS::Publisher::DATAWRITER_QOS_DEFAULT [static, get, inherited]

Special value for creating **DDS::DataWriter** (p. 499) with default QoS.

When used in **DDS::Publisher::create_datawriter** (p. 1053), this special value is used to indicate that the **DDS::DataWriter** (p. 499) should be created with the default **DDS::DataWriter** (p. 499) QoS by means of the operation get_default_datawriter_qos and using the resulting QoS to create the **DDS::DataWriter** (p. 499).

When used in **DDS::Publisher::set_default_datawriter_qos** (p. 1049), 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. 1049) operation had never been called.

When used in **DDS::DataWriter::set_qos** (p. 513), this special value is used to indicate that the QoS of the **DDS::DataWriter** (p. 499) should be changed to match the current defualt QoS set in the **DDS::Publisher** (p. 1044) that the **DDS::DataWriter** (p. 499) belongs to.

See also:

DDS::Publisher::create_datawriter (p. 1053)
DDS::Publisher::set_default_datawriter_qos (p. 1049)
DDS::DataWriter::set_qos (p. 513)

Examples:

HelloWorld_publisher.cpp.

5.13.2.2 DataWriterQos^ DDS::Publisher::DATAWRITER_QOS_USE_TOPIC_QOS [static, get, inherited]

Special value for creating **DDS::DataWriter** (p. 499) with a combination of the default **DDS::DataWriterQos** (p. 546) and the **DDS::TopicQos** (p. 1280).

The use of this value is equivalent to the application obtaining the default **DDS::DataWriterQos** (p. 546) and the **DDS::TopicQos** (p. 1280) (by means of the operation **DDS::Topic::get_qos** (p. 1262)) and then combining these two QoS using the operation **DDS::Publisher::copy_from_topic_qos** (p. 1061) whereby any policy that is set on the **DDS::TopicQos** (p. 1280) "overrides" the corresponding policy on the default QoS. The resulting QoS is then applied to the creation of the **DDS::DataWriter** (p. 499).

This value should only be used in **DDS::Publisher::create_datawriter** (p. 1053).

See also:

DDS::Publisher::create_datawriter (p. 1053)
DDS::Publisher::get_default_datawriter_qos (p. 1048)
DDS::Topic::get_qos (p. 1262)
DDS::Publisher::copy_from_topic_qos (p. 1061)

5.14 Data Writers

DDS::DataWriter (p. 499) entity and associated elements

Classes

- struct **DDS::OfferedDeadlineMissedStatus**

  DDS::StatusKind::OFFERED_DEADLINE_MISSED_STATUS

- struct **DDS::LivelinessLostStatus**

  DDS::StatusKind::LIVELINESS_LOST_STATUS

- class **DDS::OfferedIncompatibleQosStatus**

  DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS

- struct **DDS::PublicationMatchedStatus**

  DDS::StatusKind::PUBLICATION_MATCHED_STATUS

- struct **DDS::ReliableWriterCacheEventCount**

  <<**eXtension**>> (p. 174) *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. 174) *A summary of the state of a data writer's cache of unacknowledged samples written.*

- struct **DDS::ReliableReaderActivityChangedStatus**

  <<**eXtension**>> (p. 174) *Describes the activity (i.e. are acknowledgements forthcoming) of reliable readers matched to a reliable writer.*

- struct **DDS::DataWriterCacheStatus**

  <<**eXtension**>> (p. 174) *The status of the writer's cache.*

- struct **DDS::DataWriterProtocolStatus**

  <<**eXtension**>> (p. 174) *The status of a writer's internal protocol related metrics, like the number of samples pushed, pulled, filtered; and status of wire protocol traffic.*

- class **DDS::DataWriterQos**

  *QoS policies supported by a **DDS::DataWriter** (p. 499) entity.*

- class **DDS::DataWriterListener**

<<**interface**>> (p. 175) ***DDS::Listener*** *(p. 952) for writer status.*

ˆ class **DDS::DataWriter**

<<**interface**>> (p. 175) *Allows an application to set the value of the data to be published under a given* ***DDS::Topic*** *(p. 1258).*

ˆ class **DDS::TypedDataWriter< T >**

<<**interface**>> (p. 175) <<**generic**>> (p. 175) *User data type specific data writer.*

ˆ class **FooDataWriter**

<<**interface**>> (p. 175) <<**generic**>> (p. 175) *User data type specific data writer.*

5.14.1 Detailed Description

DDS::DataWriter (p. 499) entity and associated elements

5.15 Flow Controllers

<<*eXtension*>> *(p. 174)* **DDS::FlowController** (p. 867) and associated elements

Classes

^ struct **DDS::FlowControllerTokenBucketProperty_t**

DDS::FlowController *(p. 867) uses the popular token bucket approach for open loop network flow control. The flow control characteristics are determined by the token bucket properties.*

^ class **DDS::FlowControllerProperty_t**

Determines the flow control characteristics of the ***DDS::FlowController*** *(p. 867).*

^ class **DDS::FlowController**

<<**interface**>> (p. 175) *A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous* ***DDS::DataWriter*** *(p. 499) instances are allowed to write data.*

Enumerations

^ enum **DDS::FlowControllerSchedulingPolicy** {
DDS::DDS_RR_FLOW_CONTROLLER_SCHED_POLICY,
DDS::DDS_EDF_FLOW_CONTROLLER_SCHED_POLICY }

Kinds of flow controller scheduling policy.

Properties

^ static System::String^ **DDS::FlowController::DEFAULT_FLOW_CONTROLLER_NAME** `[get]`

[**default**] *Special value of* ***DDS::PublishModeQosPolicy::flow_controller_name*** *(p. 1079) that refers to the built-in default flow controller.*

^ static System::String^ **DDS::FlowController::FIXED_RATE_FLOW_CONTROLLER_NAME** `[get]`

Special value of ***DDS::PublishModeQosPolicy::flow_controller_name*** *(p. 1079) that refers to the built-in fixed-rate flow controller.*

^ static System::String^ **DDS::FlowController::ON_DEMAND_FLOW_CONTROLLER_NAME** [get]

*Special value of **DDS::PublishModeQosPolicy::flow_controller_name** (p. 1079) that refers to the built-in on-demand flow controller.*

5.15.1 Detailed Description

<<***eXtension***>> *(p. 174)* **DDS::FlowController** (p. 867) and associated elements

DDS::FlowController (p. 867) provides the network traffic shaping capability to asynchronous **DDS::DataWriter** (p. 499) instances. For use cases and advantages of publishing asnychronously, please refer to **DDS::PublishModeQosPolicy** (p. 1077) of **DDS::DataWriterQos** (p. 546).

See also:

DDS::PublishModeQosPolicy (p. 1077)
DDS::DataWriterQos::publish_mode (p. 551)
DDS::AsynchronousPublisherQosPolicy (p. 371)

5.15.2 Enumeration Type Documentation

5.15.2.1 enum DDS::FlowControllerSchedulingPolicy

Kinds of flow controller scheduling policy.

Samples written by an asynchronous **DDS::DataWriter** (p. 499) are not sent in the context of the **DDS::TypedDataWriter::write** (p. 1376) call. Instead, the middleware puts the samples in a queue for future processing. The **DDS::FlowController** (p. 867) associated with each asynchronous **DataWriter** (p. 499) instance determines when the samples are actually sent.

Each **DDS::FlowController** (p. 867) maintains a separate FIFO queue for each unique destination (remote application). Samples written by asynchronous **DDS::DataWriter** (p. 499) 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. 871)

Enumerator:

DDS_RR_FLOW_CONTROLLER_SCHED_POLICY Indicates to flow control in a round-robin fashion.

Whenever tokens become available, the flow controller distributes the tokens uniformly across all of its (non-empty) destination queues. No destinations are prioritized. Instead, all destinations are treated equally and are serviced in a round-robin fashion.

DDS_EDF_FLOW_CONTROLLER_SCHED_POLICY Indicates to flow control in an earliest-deadline-first fashion.

A sample's deadline is determined by the time it was written plus the latency budget of the **DataWriter** (p. 499) at the time of the write call (as specified in the **DDS::LatencyBudgetQosPolicy** (p. 948)). 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. 867) distributes tokens to the destination queues in order of their 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. 949) setting, an `EDF_FLOW_CONTROLLER_SCHED_POLICY` **DDS::FlowController** (p. 867) preserves the order in which the user calls **DDS::TypedDataWriter::write** (p. 1376) across the DataWriters associated with the flow controller.

Since the **DDS::LatencyBudgetQosPolicy** (p. 948) is mutable, a sample written second may contain an earlier deadline than the sample written first if the **DDS::LatencyBudgetQosPolicy::duration** (p. 949) 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. 949) and adhere to the **DDS::DataWriter** (p. 499) in-order data delivery guarantee.

[**default**] for **DDS::DataWriter** (p. 499)

5.15.3 Properties

5.15.3.1 System:: String^ DDS::FlowController::DEFAULT_FLOW_CONTROLLER_NAME [static, get, inherited]

[**default**] Special value of **DDS::PublishModeQosPolicy::flow_controller_name** (p. 1079) that refers to the built-in default flow controller.

RTI Data Distribution Service provides several built-in **DDS::FlowController** (p. 867) for use with an asynchronous **DDS::DataWriter** (p. 499). 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 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. 499).

Essentially, this is equivalent to a user-created **DDS::FlowController** (p. 867) with the following **DDS::FlowControllerProperty_t** (p. 871) settings:

- **DDS::FlowControllerProperty_t::scheduling_policy** (p. 872) = DDS::FlowControllerSchedulingPolicy::EDF_FLOW_CONTROLLER_SCHED_POLICY

- **DDS::FlowControllerProperty_t::token_bucket** (p. 872) `max_tokens` = DDS::LENGTH_UNLIMITED

- **DDS::FlowControllerProperty_t::token_bucket** (p. 872) `tokens_added_per_period` = DDS::LENGTH_UNLIMITED

- **DDS::FlowControllerProperty_t::token_bucket** (p. 872) `tokens_leaked_per_period` = 0

- **DDS::FlowControllerProperty_t::token_bucket** (p. 872) `period` = 1 second

- **DDS::FlowControllerProperty_t::token_bucket** (p. 872) `bytes_per_token` = DDS::LENGTH_UNLIMITED

See also:

DDS::Publisher::create_datawriter (p. 1053)
DDS::DomainParticipant::lookup_flowcontroller (p. 632)
DDS::FlowController::set_property (p. 868)
DDS::PublishModeQosPolicy (p. 1077)
DDS::AsynchronousPublisherQosPolicy (p. 371)

5.15.3.2 System:: String^ DDS::FlowController::FIXED_RATE_FLOW_CONTROLLER_NAME [static, get, inherited]

Special value of **DDS::PublishModeQosPolicy::flow_controller_name** (p. 1079) that refers to the built-in fixed-rate flow controller.

RTI Data Distribution Service provides several builtin **DDS::FlowController** (p. 867) for use with an asynchronous **DDS::DataWriter** (p. 499). 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 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. 867) with the following **DDS::FlowControllerProperty_t** (p. 871) settings:

- **DDS::FlowControllerProperty_t::scheduling_policy** (p. 872) = DDS::FlowControllerSchedulingPolicy::EDF_FLOW_CONTROLLER_SCHED_POLICY

- **DDS::FlowControllerProperty_t::token_bucket** (p. 872) `max_tokens` = DDS::LENGTH_UNLIMITED

- **DDS::FlowControllerProperty_t::token_bucket** (p. 872) `tokens_added_per_period` = DDS::LENGTH_UNLIMITED

- **DDS::FlowControllerProperty_t::token_bucket** (p. 872) `tokens_leaked_per_period` = DDS::LENGTH_UNLIMITED

- **DDS::FlowControllerProperty_t::token_bucket** (p. 872) `period` = 1 second

- **DDS::FlowControllerProperty_t::token_bucket** (p. 872) `bytes_per_token` = DDS::LENGTH_UNLIMITED

See also:

DDS::Publisher::create_datawriter (p. 1053)
DDS::DomainParticipant::lookup_flowcontroller (p. 632)
DDS::FlowController::set_property (p. 868)
DDS::PublishModeQosPolicy (p. 1077)
DDS::AsynchronousPublisherQosPolicy (p. 371)

5.15.3.3 System:: String^ DDS::FlowController::ON_DEMAND_FLOW_CONTROLLER_NAME [static, get, inherited]

Special value of **DDS::PublishModeQosPolicy::flow_controller_name** (p. 1079) that refers to the built-in on-demand flow controller.

RTI Data Distribution Service provides several builtin **DDS::FlowController** (p. 867) for use with an asynchronous **DDS::DataWriter** (p. 499). 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 allows data to be sent only when the user calls **DDS::FlowController::trigger_flow** (p. 869). With each trigger, all accumulated data since the previous trigger is sent (across all **DDS::Publisher** (p. 1044) or **DDS::DataWriter** (p. 499) 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 DDS::Transport_Property_t::gather_send_buffer_count_max).

Essentially, this is equivalent to a user-created **DDS::FlowController** (p. 867) with the following **DDS::FlowControllerProperty_t** (p. 871) settings:

- **DDS::FlowControllerProperty_t::scheduling_policy** (p. 872) = DDS::FlowControllerSchedulingPolicy::EDF_FLOW_CONTROLLER_SCHED_POLICY

- **DDS::FlowControllerProperty_t::token_bucket** (p. 872) `max_tokens` = DDS::LENGTH_UNLIMITED

- **DDS::FlowControllerProperty_t::token_bucket** (p. 872) `tokens_added_per_period` = DDS::LENGTH_UNLIMITED

- **DDS::FlowControllerProperty_t::token_bucket** (p. 872) `tokens_leaked_per_period` = DDS::LENGTH_UNLIMITED

- **DDS::FlowControllerProperty_t::token_bucket** (p. 872) `period` = **DDS::Duration_t::DURATION_INFINITE** (p. 253)

- **DDS::FlowControllerProperty_t::token_bucket** (p. 872) `bytes_per_token` = DDS::LENGTH_UNLIMITED

See also:

DDS::Publisher::create_datawriter (p. 1053)
DDS::DomainParticipant::lookup_flowcontroller (p. 632)
DDS::FlowController::trigger_flow (p. 869)

DDS::FlowController::set_property (p. 868)
DDS::PublishModeQosPolicy (p. 1077)
DDS::AsynchronousPublisherQosPolicy (p. 371)

5.16 Subscription Module

Contains the **DDS::Subscriber** (p. 1201), **DDS::DataReader** (p. 433), **DDS::ReadCondition** (p. 1084), and **DDS::QueryCondition** (p. 1082) classes, as well as the **DDS::SubscriberListener** (p. 1226) and **DDS::DataReaderListener** (p. 461) interfaces, and more generally, all that is needed on the subscription side.

Modules

- **Subscribers**

  DDS::Subscriber *(p. 1201) entity and associated elements*

- **DataReaders**

  DDS::DataReader *(p. 433) entity and associated elements*

- **Data Samples**

  DDS::SampleInfo *(p. 1148),* ***DDS::SampleStateKind*** *(p. 1161),* ***DDS::ViewStateKind*** *(p. 1409),* ***DDS::InstanceStateKind*** *(p. 907) and associated elements*

5.16.1 Detailed Description

Contains the **DDS::Subscriber** (p. 1201), **DDS::DataReader** (p. 433), **DDS::ReadCondition** (p. 1084), and **DDS::QueryCondition** (p. 1082) classes, as well as the **DDS::SubscriberListener** (p. 1226) and **DDS::DataReaderListener** (p. 461) interfaces, and more generally, all that is needed on the subscription side.

5.16.2 Access to data samples

Data is made available to the application by the following operations on **DDS::DataReader** (p. 433) objects: **DDS::TypedDataReader::read** (p. 1341), **DDS::TypedDataReader::read_w_condition** (p. 1349), **DDS::TypedDataReader::take** (p. 1342), **DDS::TypedDataReader::take_w_condition** (p. 1350), 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 Data Distribution Service 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 Data Distribution Service. 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. 1148) 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. 294) QoS allow for it.

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 DDS::TypedDataReader::return_loan (p. 1364).

See also:

Interpretation of the SampleInfo (p. 1149)

5.16.2.1 Data access patterns

The application accesses data by means of the operations `read` or `take` on the **DDS::DataReader** (p. 433). These operations return an ordered collection of `DataSamples` consisting of a **DDS::SampleInfo** (p. 1148) part and a `Data` part.

The way RTI Data Distribution Service builds the collection depends on QoS policies set on the **DDS::DataReader** (p. 433) and **DDS::Subscriber** (p. 1201), 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. 1161))
- the desired view states (any combination of **DDS::ViewStateKind** (p. 1409))
- the desired instance states (any combination of **DDS::InstanceStateKind** (p. 907))

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. 1084) 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 true. These operations, in conjunction with **DDS::ReadCondition** (p. 1084) objects and a **DDS::WaitSet** (p. 1411), 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. 279) QoS must be set properly.

5.17 Subscribers

DDS::Subscriber (p. 1201) entity and associated elements

Classes

- class **DDS::SubscriberQos**

  *QoS policies supported by a **DDS::Subscriber** (p. 1201) entity.*

- class **DDS::SubscriberListener**

  <<**interface**>> (p. 175) ***DDS::Listener*** *(p. 952) for status about a subscriber.*

- class **DDS::SubscriberSeq**

  Declares IDL `sequence` *<* ***DDS::Subscriber*** *(p. 1201)* *>* .

- class **DDS::Subscriber**

  <<**interface**>> (p. 175) *A subscriber is the object responsible for actually receiving data from a subscription.*

Properties

- static DataReaderQos^ **DDS::Subscriber::DATAREADER_QOS_DEFAULT** `[get]`

  Special value for creating data reader with default QoS.

- static DataReaderQos^ **DDS::Subscriber::DATAREADER_QOS_USE_TOPIC_QOS** `[get]`

  *Special value for creating **DDS::DataReader** (p. 433) with a combination of the default **DDS::DataReaderQos** (p. 480) and the **DDS::TopicQos** (p. 1280).*

5.17.1 Detailed Description

DDS::Subscriber (p. 1201) entity and associated elements

5.17.2 Properties

5.17.2.1 DataReaderQos^ DDS::Subscriber::DATAREADER_QOS_DEFAULT [static, get, inherited]

Special value for creating data reader with default QoS.

When used in **DDS::Subscriber::create_datareader** (p. 1210), this special value is used to indicate that the **DDS::DataReader** (p. 433) should be created with the default **DDS::DataReader** (p. 433) QoS by means of the operation get_default_datareader_qos and using the resulting QoS to create the **DDS::DataReader** (p. 433).

When used in **DDS::Subscriber::set_default_datareader_qos** (p. 1206), 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. 1206) operation had never been called.

When used in **DDS::DataReader::set_qos** (p. 448), this special value is used to indicate that the QoS of the **DDS::DataReader** (p. 433) should be changed to match the current defualt QoS set in the **DDS::Subscriber** (p. 1201) that the **DDS::DataReader** (p. 433) belongs to.

See also:

DDS::Subscriber::create_datareader (p. 1210)
DDS::Subscriber::set_default_datareader_qos (p. 1206)
DDS::DataReader::set_qos (p. 448)

Examples:

HelloWorld_subscriber.cpp.

5.17.2.2 DataReaderQos^ DDS::Subscriber::DATAREADER_QOS_USE_TOPIC_QOS [static, get, inherited]

Special value for creating **DDS::DataReader** (p. 433) with a combination of the default **DDS::DataReaderQos** (p. 480) and the **DDS::TopicQos** (p. 1280).

The use of this value is equivalent to the application obtaining the default **DDS::DataReaderQos** (p. 480) and the **DDS::TopicQos** (p. 1280) (by means of the operation **DDS::Topic::get_qos** (p. 1262)) and then combining these two QoS using the operation **DDS::Subscriber::copy_from_topic_qos** (p. 1220) whereby any policy that is set on the **DDS::TopicQos** (p. 1280) "overrides" the corresponding policy on the default QoS. The resulting QoS is then applied to the creation of the **DDS::DataReader** (p. 433).

This value should only be used in **DDS::Subscriber::create_datareader** (p. 1210).

See also:

DDS::Subscriber::create_datareader (p. 1210)
DDS::Subscriber::get_default_datareader_qos (p. 1205)
DDS::Topic::get_qos (p. 1262)
DDS::Subscriber::copy_from_topic_qos (p. 1220)

5.18 DataReaders

DDS::DataReader (p. 433) entity and associated elements

Modules

- **Read Conditions**

  DDS::ReadCondition *(p. 1084) and associated elements*

- **Query Conditions**

  DDS::QueryCondition *(p. 1082) and associated elements*

Classes

- struct **DDS::RequestedDeadlineMissedStatus**

  DDS::StatusKind::REQUESTED_DEADLINE_MISSED_STATUS

- struct **DDS::LivelinessChangedStatus**

  DDS::StatusKind::LIVELINESS_CHANGED_STATUS

- class **DDS::RequestedIncompatibleQosStatus**

  DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS

- struct **DDS::SampleLostStatus**

  DDS::StatusKind::SAMPLE_LOST_STATUS

- struct **DDS::SampleRejectedStatus**

  DDS::StatusKind::SAMPLE_REJECTED_STATUS

- struct **DDS::SubscriptionMatchedStatus**

  DDS::StatusKind::SUBSCRIPTION_MATCHED_STATUS

- struct **DDS::DataReaderCacheStatus**

  <<**eXtension**>> (p. 174) *The status of the reader's cache.*

- struct **DDS::DataReaderProtocolStatus**

  <<**eXtension**>> (p. 174) *The status of a reader's internal protocol related metrics, like the number of samples received, filtered, rejected; and status of wire protocol traffic.*

- class **DDS::DataReaderQos**

*QoS policies supported by a **DDS::DataReader*** (p. *433*) *entity.*

ˆ class **DDS::DataReaderSeq**

Declares IDL `sequence` *<* ***DDS::DataReader*** (p. *433*) *>* .

ˆ class **DDS::DataReaderListener**

<<**interface**>> (p. 175) ***DDS::Listener*** (p. *952*) *for reader status.*

ˆ class **DDS::DataReader**

<<**interface**>> (p. 175) *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. *1201*).

ˆ class **DDS::TypedDataReader< T >**

<<**interface**>> (p. 175) <<**generic**>> (p. 175) *User data type-specific data reader.*

ˆ class **FooDataReader**

<<**interface**>> (p. 175) <<**generic**>> (p. 175) *User data type-specific data reader.*

Enumerations

ˆ enum **DDS::SampleRejectedStatusKind** {

DDS::NOT_REJECTED,

DDS::REJECTED_BY_INSTANCES_LIMIT,

DDS::REJECTED_BY_SAMPLES_LIMIT,

DDS::REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT,

DDS::REJECTED_BY_REMOTE_WRITERS_LIMIT,

DDS::REJECTED_BY_REMOTE_WRITERS_PER_-INSTANCE_LIMIT,

DDS::REJECTED_BY_SAMPLES_PER_REMOTE_WRITER_-LIMIT }

Kinds of reasons for rejecting a sample.

5.18.1 Detailed Description

DDS::DataReader (p. 433) entity and associated elements

5.18.2 Enumeration Type Documentation

5.18.2.1 enum DDS::SampleRejectedStatusKind

Kinds of reasons for rejecting a sample.

Enumerator:

NOT_REJECTED Samples are never rejected.
See also:

ResourceLimitsQosPolicy (p. 1109)

REJECTED_BY_INSTANCES_LIMIT A resource limit on the number of instances was reached.
See also:

ResourceLimitsQosPolicy (p. 1109)

REJECTED_BY_SAMPLES_LIMIT A resource limit on the number of samples was reached.
See also:

ResourceLimitsQosPolicy (p. 1109)

REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT A resource limit on the number of samples per instance was reached.
See also:

ResourceLimitsQosPolicy (p. 1109)

REJECTED_BY_REMOTE_WRITERS_LIMIT A resource limit on the number of remote writers from which a **DDS::DataReader** (p. 433) may read was reached.
This constant is an extension to the DDS standard.
See also:

DDS::DataReaderResourceLimitsQosPolicy (p. 486)

REJECTED_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. 433) may read was reached.
This constant is an extension to the DDS standard.
See also:

DDS::DataReaderResourceLimitsQosPolicy (p. 486)

REJECTED_BY_SAMPLES_PER_REMOTE_WRITER_LIMIT A resource limit on the number of samples from a given remote writer that a **DDS::DataReader** (p. 433) may store was reached.
This constant is an extension to the DDS standard.
See also:

DDS::DataReaderResourceLimitsQosPolicy (p. 486)

5.19 Read Conditions

DDS::ReadCondition (p. 1084) and associated elements

Classes

ˆ class **DDS::ReadCondition**

<<**interface**>> (p. 175) *Conditions specifically dedicated to read operations and attached to one* ***DDS::DataReader*** *(p. 433).*

5.19.1 Detailed Description

DDS::ReadCondition (p. 1084) and associated elements

5.20 Query Conditions

DDS::QueryCondition (p. 1082) and associated elements

Classes

- ˆ class **DDS::QueryCondition**

  <<**interface**>> (p. 175) *These are specialised* ***DDS::ReadCondition*** *(p. 1084) objects that allow the application to also specify a filter on the locally available data.*

5.20.1 Detailed Description

DDS::QueryCondition (p. 1082) and associated elements

5.21 Data Samples

DDS::SampleInfo (p. 1148), **DDS::SampleStateKind** (p. 1161), **DDS::ViewStateKind** (p. 1409), **DDS::InstanceStateKind** (p. 907) and associated elements

Modules

- **Sample States**

  DDS::SampleStateKind *(p. 1161) and associated elements*

- **View States**

  DDS::ViewStateKind *(p. 1409) and associated elements*

- **Instance States**

  DDS::InstanceStateKind *(p. 907) and associated elements*

Classes

- class **DDS::SampleInfo**

  Information that accompanies each sample that is `read` *or* `taken`.

- class **DDS::SampleInfoSeq**

  Declares IDL `sequence` < ***DDS::SampleInfo*** *(p. 1148)* > .

5.21.1 Detailed Description

DDS::SampleInfo (p. 1148), **DDS::SampleStateKind** (p. 1161), **DDS::ViewStateKind** (p. 1409), **DDS::InstanceStateKind** (p. 907) and associated elements

5.22 Sample States

DDS::SampleStateKind (p. 1161) and associated elements

Classes

- ^ struct **DDS::SampleStateKind**

  Indicates whether or not a sample has ever been read.

Properties

- ^ static SampleStateKind **DDS::SampleStateKind::ANY_SAMPLE_STATE** `[get]`

  Any sample state ***DDS::SampleStateKind::READ_SAMPLE_STATE*** *(p. 1161)* | ***DDS::SampleStateKind::NOT_READ_SAMPLE_STATE*** *(p. 1162).*

5.22.1 Detailed Description

DDS::SampleStateKind (p. 1161) and associated elements

5.22.2 Properties

5.22.2.1 SampleStateKind DDS::SampleStateKind::ANY_SAMPLE_STATE `[static, get, inherited]`

Any sample state **DDS::SampleStateKind::READ_SAMPLE_STATE** (p. 1161) | **DDS::SampleStateKind::NOT_READ_SAMPLE_STATE** (p. 1162).

Examples:

HelloWorld_subscriber.cpp.

5.23 View States

DDS::ViewStateKind (p. 1409) and associated elements

Classes

- struct **DDS::ViewStateKind**

  Indicates whether or not an instance is new.

Properties

- static ViewStateKind **DDS::ViewStateKind::ANY_VIEW_STATE** `[get]`

  Any view state ***DDS::ViewStateKind::NEW_VIEW_STATE*** *(p. 1410)* | ***DDS::ViewStateKind::NOT_NEW_VIEW_STATE*** *(p. 1410).*

5.23.1 Detailed Description

DDS::ViewStateKind (p. 1409) and associated elements

5.23.2 Properties

5.23.2.1 ViewStateKind DDS::ViewStateKind::ANY_VIEW_STATE `[static, get, inherited]`

Any view state **DDS::ViewStateKind::NEW_VIEW_STATE** (p. 1410) | **DDS::ViewStateKind::NOT_NEW_VIEW_STATE** (p. 1410).

Examples:

HelloWorld_subscriber.cpp.

5.24 Instance States

DDS::InstanceStateKind (p. 907) and associated elements

Classes

- struct **DDS::InstanceStateKind**

  *Indicates is the samples are from a live **DDS::DataWriter** (p. 499) or not.*

Properties

- static InstanceStateKind **DDS::InstanceStateKind::ANY_INSTANCE_STATE** [get]

  Any instance state ALIVE_INSTANCE_STATE | NOT_ALIVE_DISPOSED_INSTANCE_STATE | NOT_ALIVE_NO_WRITERS_INSTANCE_STATE.

- static InstanceStateKind **DDS::InstanceStateKind::NOT_ALIVE_INSTANCE_STATE** [get]

  Not alive instance state NOT_ALIVE_DISPOSED_INSTANCE_STATE | NOT_ALIVE_NO_WRITERS_INSTANCE_STATE.

5.24.1 Detailed Description

DDS::InstanceStateKind (p. 907) and associated elements

5.24.2 Properties

5.24.2.1 InstanceStateKind DDS::InstanceStateKind::ANY_INSTANCE_STATE [static, get, inherited]

Any instance state ALIVE_INSTANCE_STATE | NOT_ALIVE_DISPOSED_INSTANCE_STATE | NOT_ALIVE_NO_WRITERS_INSTANCE_STATE.

Examples:

HelloWorld_subscriber.cpp.

5.24.2.2 InstanceStateKind DDS::InstanceStateKind::NOT_ALIVE_INSTANCE_STATE [static, get, inherited]

Not alive instance state NOT_ALIVE_DISPOSED_INSTANCE_STATE | NOT_ALIVE_NO_WRITERS_INSTANCE_STATE.

5.25 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

- **Return Codes**

  Types of return codes.

- **Status Kinds**

  Kinds of communication status.

- **Exception Codes**

  <<**eXtension**>> (p. 174) ***Exception*** *(p. 861) codes.*

- **Time Support**

  Time and duration types and defines.

- **GUID Support**

  <<**eXtension**>> (p. 174) *GUID type and defines.*

- **Sequence Number Support**

  <<**eXtension**>> (p. 174) ***Sequence*** *(p. 1163) number type and defines.*

- **QoS Policies**

  Quality of Service (QoS) policies.

- **Entity Support**

  DDS::Entity *(p. 845),* ***DDS::Listener*** *(p. 952) and related items.*

- **Conditions and WaitSets**

  DDS::Condition *(p. 408) and* ***DDS::WaitSet*** *(p. 1411) and related items.*

- **Sequence Support**

  The ***DDS::Sequence*** *(p. 1163) interface allows you to work with variable-length collections of homogeneous data.*

5.25.1 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.

5.26 Built-in Sequences

Defines sequences of primitive data type.

Classes

- ^ class **DDS::StringSeq**

  Instantiates `DDS::Sequence` (p. 1163) < *System::String* > *with value type semantics.*

- ^ class **DDS::WstringSeq**

  Instantiates `DDS::Sequence` (p. 1163) < *System::Char*∗ >.

- ^ class **DDS::CharSeq**

  Instantiates `DDS::Sequence` (p. 1163) < *System::Char* >.

- ^ class **DDS::WcharSeq**

  Instantiates `DDS::Sequence` (p. 1163) < *System::Char* >.

- ^ class **DDS::ByteSeq**

  Instantiates `DDS::Sequence` (p. 1163) < *System::Byte* >.

- ^ class **DDS::ShortSeq**

  Instantiates `DDS::Sequence` (p. 1163) < *System::Int16* >.

- ^ class **DDS::UnsignedShortSeq**

  Instantiates `DDS::Sequence` (p. 1163) < *System::UInt16* >.

- ^ class **DDS::IntSeq**

  Instantiates `DDS::Sequence` (p. 1163) < *System::Int32* >.

- ^ class **DDS::UnsignedIntSeq**

  Instantiates `DDS::Sequence` (p. 1163) < *System::UInt32* >.

- ^ class **DDS::LongSeq**

  Instantiates `DDS::Sequence` (p. 1163) < *System::Int64* >.

- ^ class **DDS::UnsignedLongSeq**

  Instantiates `DDS::Sequence` (p. 1163) < *System::UInt64* >.

- ^ class **DDS::FloatSeq**

  Instantiates `DDS::Sequence` (p. 1163) < *System::Single* >.

- class **DDS::DoubleSeq**

  Instantiates `DDS::Sequence` (p. 1163) < *System::Double* >.

- class **DDS::LongDoubleSeq**

  Instantiates `DDS::Sequence` (p. 1163) < ***DDS::LongDouble*** *(p. 976)* >.

- class **DDS::BooleanSeq**

  Instantiates `DDS::Sequence` (p. 1163) < *System::Boolean* >.

5.26.1 Detailed Description

Defines sequences of primitive data type.

5.27 Multi-channel DataWriters

APIs related to Multi-channel DataWriters.

5.27.1 What is a Multi-channel DataWriter?

A Multi-channel **DDS::DataWriter** (p. 499) is a **DDS::DataWriter** (p. 499) 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. 499). Each filter "guards" a channel (a set of multicast addresses). Each time a multi-channel **DDS::DataWriter** (p. 499) 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.

5.27.2 Configuration on the Writer Side

To configure a multi-channel **DDS::DataWriter** (p. 499), simply define a list of all its channels in the **DDS::MultiChannelQosPolicy** (p. 981).

The **DDS::MultiChannelQosPolicy** (p. 981) is propagated along with discovery traffic. The value of this policy is available in **DDS::PublicationBuiltinTopicData::locator_filter** (p. 1035).

5.27.3 Configuration on the Reader Side

No special changes are required in a subscribing application to get data from a multichannel **DDS::DataWriter** (p. 499). If you want the **DDS::DataReader** (p. 433) to subscribe to only a subset of the channels, use a **DDS::ContentFilteredTopic** (p. 419).

For more information on Multi-channel DataWriters, refer to the User's Manual.

5.27.4 Reliability with Multi-Channel DataWriters

5.27.4.1 Reliable Delivery

Reliable delivery is only guaranteed when the **DDS::PresentationQosPolicy::access_scope** (p. 1015) is set to DDS::PresentationQosPolicyAccessScopeKind::INSTANCE_-

PRESENTATION_QOS and the filters in **DDS::MultiChannelQosPolicy** (p. 981) are keyed-only based.

If any of the guard filters are based on non-key fields, RTI Data Distribution Service only guarantees reception of the most recent data from the MultiChannel DataWriter.

5.27.4.2 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. 1112) and/or **DDS::DataWriterResourceLimitsQosPolicy::max_batches** (p. 555).

The protocol parameters described in **DDS::DataWriterProtocolQosPolicy** (p. 529) are applied per channel, with the following exceptions:

DDS::RtpsReliableWriterProtocol_t::low_watermark (p. 1130) and **DDS::RtpsReliableWriterProtocol_t::high_watermark** (p. 1131): 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. 1133): 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. 1112))

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. 1112) instead of max_batches **DDS::DataWriterResourceLimitsQosPolicy::max_batches** (p. 555) in order to take advantage of heartbeats_per_max_samples.

5.28 Pluggable Transports

APIs related to RTI Data Distribution Service pluggable transports.

Modules

- **Using Transport Plugins**

  Configuring transports used by RTI Data Distribution Service.

- **Built-in Transport Plugins**

  Transport plugins delivered with RTI Data Distribution Service.

5.28.1 Detailed Description

APIs related to RTI Data Distribution Service pluggable transports.

5.28.2 Overview

RTI Data Distribution Service has a pluggable transports architecture. The core of RTI Data Distribution Service is transport agnostic; it does not make any assumptions about the actual transports used to send and receive messages. Instead, the RTI Data Distribution Service 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. 122)) is delivered with RTI Data Distribution Service for commonly used transports. New transport plugins can easily be created, thus enabling RTI Data Distribution Service applications to run over transports that may not even be conceived yet. This is a powerful capability and that distinguishes RTI Data Distribution Service from competing middleware approaches.

RTI Data Distribution Service also provides a set of APIs for installing and configuring transport plugins to be used in an application. So that RTI Data Distribution Service applications work out of the box, a subset of the builtin transport plugins is *preconfigured* by default (see **DDS::TransportBuiltinQosPolicy** (p. 1285)). 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.

5.28.3 Transport Aliases

In order to use a transport plugin instance in an RTI Data Distribution Service application, it must be registered with a **DDS::DomainParticipant** (p. 577). 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. 577). An **alias** symbolically refers to one or more transport plugins registered with the **DDS::DomainParticipant** (p. 577). Builtin transport plugin instances can be referred to using preconfigured aliases (see **TRANSPORT_BUILTIN** (p. 321)).

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. 572)), and for **DDS::DataWriter** (p. 499) and **DDS::DataReader** (p. 433) entities (see **DDS::TransportSelectionQosPolicy** (p. 1294)).

- the **multicast** addresses on which to receive discovery messages (see **DDS::DiscoveryQosPolicy::multicast_receive_addresses** (p. 573)), and the multicast addresses and ports on which to receive user data (see **DDS::DataReaderQos::multicast** (p. 484)).

- the **unicast ports** used for user data (see **DDS::TransportUnicastQosPolicy** (p. 1296)) on both **DDS::DataWriter** (p. 499) and **DDS::DataReader** (p. 433) entities.

- the transport plugins used to parse an address string in a locator (**Locator Format** (p. 313) and **NDDS_DISCOVERY_PEERS** (p. 312)).

A **DDS::DomainParticipant** (p. 577) (and contained its entities) start using a transport plugin after the **DDS::DomainParticipant** (p. 577) is enabled (see **DDS::Entity::enable** (p. 848)). An entity will use *all* the transport plugins that match the specified transport QoS policy. All transport plugins are treated uniformly, regardless of how they were created or registered; there is no notion of some transports being more "special" that others.

5.28.4 Transport Lifecycle

A transport plugin is owned by whoever created it. Thus, if you create and register a transport plugin with a **DDS::DomainParticipant** (p. 577), you are responsible for deleting it by calling its destructor. Note that builtin transport plugins (**TRANSPORT_BUILTIN** (p. 321)) and transport plugins that are

loaded through the **PROPERTY** (p. 357) QoS policy (see **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 119)) are automatically managed by RTI Data Distribution Service.

A user-created transport plugin must not be deleted while it is still in use by a **DDS::DomainParticipant** (p. 577). This generally means that a user-created transport plugin instance can only be deleted after the **DDS::DomainParticipant** (p. 577) with which it was registered is deleted (see **DDS::DomainParticipantFactory::delete_participant** (p. 668)). Note that a transport plugin *cannot* be "unregistered" from a **DDS::DomainParticipant** (p. 577).

A transport plugin instance cannot be registered with more than one **DDS::DomainParticipant** (p. 577) 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. 577) 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. 577) 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.

5.28.5 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 DDS::Transport_Property_t::classid)

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 Data Distribution Service Core) to communicate with one another.

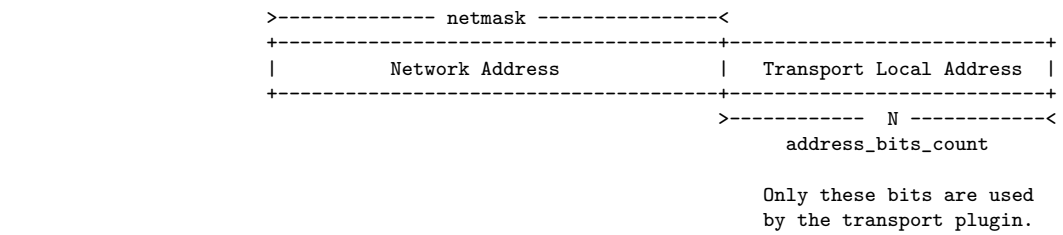
Multiple implementations (possibly from different vendors) for a physical transport mechanism can co-exist in an RTI Data Distribution Service application, provided they use different transport class IDs.

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 DDS::Transport_Property_t::address_bit_count

RTI Data Distribution Service's addressing is modeled after the IPv6 and uses 128-bit addresses (strings) to route messages.

A transport plugin is expected to map the transport's internal addressing scheme to 128-bit addresses. In general, this mapping is likely to use only `N` least significant bits (LSB); these are specified by this attribute.



The remaining bits of an address using the 128-bit address representation will be considered as part of the "network address" (see **Transport Network Address** (p. 117)) and thus ignored by the transport plugin's internal addressing scheme.

For *unicast* addresses, the transport plugin is expected to ignore the higher (128 - DDS::Transport_Property_t::address_bit_count) bits. RTI Data Distribution Service is free to manipulate those bits freely in the addresses passed in/out to the transport plugin APIs.

Theoretically, the significant address bits count, `N` is related to the size of the underlying transport network as follows:

$$address\_bits\_count >= ceil(log_2(total\_addressable\_transport\_unicast\_interfaces))$$

The equality holds when the most compact (theoretical) internal address mapping scheme is used. A practical address mapping scheme may waste some bits.

5.28.6 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 Data Distribution Service requires that every transport plugin instance must specify the DDS::Transport_Property_t::message_size_max and DDS::Transport_Property_t::gather_send_buffer_count_max.

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.

5.28.7 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 Data Distribution Service to properly route the messages, each unicast interface in the RTI Data Distribution Service *domain* must have a unique address. RTI Data Distribution Service allows the user to specify the value of the network address when installing a transport plugin via the DDS::Transport_Support::register_transport() API.

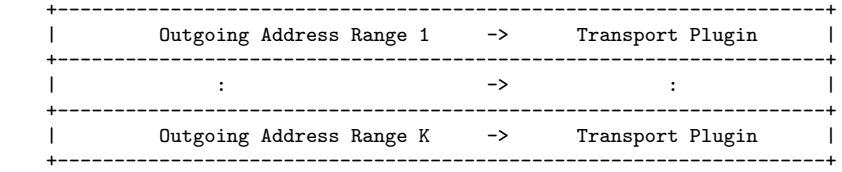
The network address for a transport plugin should be chosen such that the resulting fully qualified 128-bit address will be unique in the RTI Data Distribution Service domain. Thus, if two instances of a transport plugin are registered with a **DDS::DomainParticipant** (p. 577), they will be at different 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).

5.28.8 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 Data Distribution Service allows the user to configure the routing of outgoing messages via the DDS::Transport_Support::add_send_route() API, so that

a transport plugin will be used to send messages only to certain ranges of destination addresses. The method can be called multiple times for a transport plugin, with different address ranges.

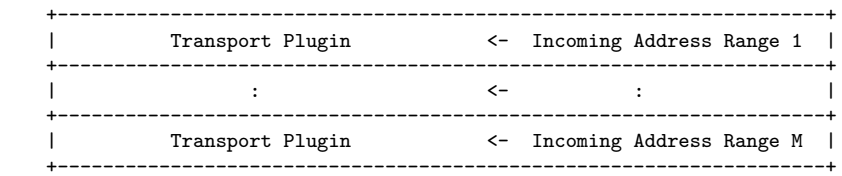


The user can set up a routing table to restrict the use of a transport plugin to send messages to selected addresses ranges.

5.28.9 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 Data Distribution Service allows the user to configure the routing of incoming messages via the DDS::Transport_Support::add_receive_route() API, so that a transport plugin will be used to receive messages only on certain ranges of addresses. The method can be called multiple times for a transport plugin, with different address ranges.



The user can set up a routing table to restrict the use of a transport plugin to receive messages from selected ranges. For example, the user may restrict a transport plugin to

- receive messages from a certain multicast address range.

- receive messages only on certain unicast interfaces (when multiple unicast interfaces are available on the transport plugin).

5.29 Using Transport Plugins

Configuring transports used by RTI Data Distribution Service. There is more than one way to install a transport plugin for use with RTI Data Distribution Service:

- If it is a builtin transport plugin, by specifying a bitmask in **DDS::TransportBuiltinQosPolicy** (p. 1285) (see **Built-in Transport Plugins** (p. 122))
- For all other non-builtin transport plugins, by dynamically loading the plugin through **PROPERTY** (p. 357) QoS policy settings of **DDS::DomainParticipant** (p. 577) (on UNIX, Solaris and Windows systems only) (see **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 119))

The lifecycle of the transport plugin is automatically managed by RTI Data Distribution Service. See **Transport Lifecycle** (p. 114) for details.

5.29.1 Loading Transport Plugins through Property QoS Policy of Domain Participant

On UNIX, Solaris and Windows operating systems, a non-builtin transport plugin written in C/C++ and built as a dynamic-link library (∗.dll/∗.so) can be loaded by RTI Data Distribution Service through the **PROPERTY** (p. 357) QoS policy settings of the **DDS::DomainParticipant** (p. 577). The dynamic-link library (and all the dependent libraries) need to be in the path during runtime (in **LD_LIBRARY_PATH** environment variable on Linux/Solaris systems,and in **PATH** environment variable for Windows systems).

To allow dynamic loading of the transport plugin, the transport plugin must implement the RTI Data Distribution Service abstract transport API and must provide a function with the signature DDS::Transport_create_plugin that can be called by RTI Data Distribution Service 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. 357) QoS policy of the **DDS::DomainParticipant** (p. 577).

The following table lists the property names that are used to load the transport plugins dynamically:

A transport plugin is dynamically created and registered to the **DDS::DomainParticipant** (p. 577) by RTI Data Distribution Service when:

- the **DDS::DomainParticipant** (p. 577) is enabled,
- the first DataWriter/DataReader is created, or
- you lookup a builtin DataReader (**DDS::Subscriber::lookup_datareader** (p. 1215)),

whichever happens first.

Any changes to the transport plugin related properties in **PROPERTY** (p. 357) QoS policy after the transport plugin has been registered with the **DDS::DomainParticipant** (p. 577) will have no effect.

Property Name	Description	Required?
dds.transport.load_plugins	Comma-separated strings indicating the prefix names of all plugins that will be loaded by RTI Data Distribution Service. Up to 8 plugins may be specified. For example, "dds.transport.WAN.wan1, dds.transport.DTLS.dtls1", In the following examples, <TRANSPORT_PREFIX> is used to indicate one element of this string that is used as a prefix in the property names for all the settings that are related to the plugin. <TRANSPORT_PREFIX> must begin with "dds.transport." (such as "dds.transport.mytransport").	**YES**
<TRANSPORT_PREFIX>.library	Should be set to the name of the dynamic library (*.so for Unix/Solaris, and *.dll for Windows) that contains the transport plugin implementation. This library (and all the other dependent dynamic libraries) needs to be in the path during run time for used by RTI Data Distribution Service (in the **LD_LIBRARY_PATH** environment variable on UNIX/Solaris systems, in **PATH** for Windows systems).	**YES**
<TRANSPORT_PREFIX>.create_function	Should be set to the name of the function with the prototype of DDS::Transport_create_plugin that can be called by RTI Data Distribution Service to create an instance of the plugin. The resulting transport	**YES**

5.30 Built-in Transport Plugins

Transport plugins delivered with RTI Data Distribution Service.

Classes

- interface **DDS::ShmemTransport**

  Built-in transport plug-in for inter-process communications using shared memory.

- interface **DDS::UDPv4Transport**

  Built-in transport plug-in using UDP/IPv4.

- interface **DDS::UDPv6Transport**

  Built-in transport plug-in using UDP/IPv6.

5.30.1 Detailed Description

Transport plugins delivered with RTI Data Distribution Service.

The **TRANSPORT_BUILTIN** (p. 321) specifies the collection of transport plugins that can be automatically configured and managed by RTI Data Distribution Service as a convenience to the user.

These transport plugins can simply be turned "on" or "off" by a specifying a bitmask in **DDS::TransportBuiltinQosPolicy** (p. 1285), thus bypassing the steps for setting up a transport plugin. RTI Data Distribution Service 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. 1285), the plugin is implicitly created and registered to the corresponding **DDS::DomainParticipant** (p. 577) by RTI Data Distribution Service when:

- the **DDS::DomainParticipant** (p. 577) is enabled,
- the first DataWriter/DataReader is created, or
- you lookup a builtin DataReader (**DDS::Subscriber::lookup_datareader** (p. 1215)),

whichever happens first.

Each builtin transport contains its own set of properties. For example, the **DDS::UDPv4Transport** (p. 1388) allows the application to specify whether or not multicast is supported, the maximum size of the message, and provides a mechanism for the application to filter out network interfaces.

The builtin transport plugin properties can be changed by the method DDS::Transport_Support::set_builtin_transport_property() or by using the **PROPERTY** (p. 357) QoS policy associated with the **DDS::DomainParticipant** (p. 577). Builtin transport plugin properties specified in **DDS::PropertyQosPolicy** (p. 1023) always overwrite the ones specified through DDS::Transport_Support::set_builtin_transport_property(). Refer to the specific builtin transport for the list of property names that can be specified through **PROPERTY** (p. 357) QoS policy.

Any changes to the builtin transport properties after the builtin transports have been registered with will have no effect.

See also:

DDS::Transport_Support::set_builtin_transport_property()
DDS::PropertyQosPolicy (p. 1023)

5.31 Configuration Utilities

Utility API's independent of the DDS standard.

Classes

- struct **NDDS::Config_LibraryVersion_t**

  The version of a single library shipped as part of an RTI Data Distribution Service distribution.

- class **NDDS::ConfigVersion**

  <<**interface**>> (p. 175) *The version of an RTI Data Distribution Service distribution.*

- class **NDDS::ConfigLogger**

  <<**interface**>> (p. 175) *The singleton type used to configure RTI Data Distribution Service logging.*

Enumerations

- enum **NDDS::LogVerbosity** {

  NDDS::NDDS_CONFIG_LOG_VERBOSITY_SILENT,

  NDDS::NDDS_CONFIG_LOG_VERBOSITY_ERROR,

  NDDS::NDDS_CONFIG_LOG_VERBOSITY_WARNING,

  NDDS::NDDS_CONFIG_LOG_VERBOSITY_STATUS_-LOCAL,

  NDDS::NDDS_CONFIG_LOG_VERBOSITY_STATUS_-REMOTE,

  NDDS::NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL }

  The verbosities at which RTI Data Distribution Service diagnostic information is logged.

- enum **NDDS::LogCategory** {

  NDDS::NDDS_CONFIG_LOG_CATEGORY_PLATFORM,

  NDDS::NDDS_CONFIG_LOG_CATEGORY_-COMMUNICATION,

  NDDS::NDDS_CONFIG_LOG_CATEGORY_DATABASE,

  NDDS::NDDS_CONFIG_LOG_CATEGORY_ENTITIES,

  NDDS::NDDS_CONFIG_LOG_CATEGORY_API }

Categories of logged messages.

^ enum **NDDS::LogPrintFormat** { ,

NDDS::NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT,

NDDS::NDDS_CONFIG_LOG_PRINT_FORMAT_-TIMESTAMPED,

NDDS::NDDS_CONFIG_LOG_PRINT_FORMAT_VERBOSE,

NDDS::NDDS_CONFIG_LOG_PRINT_FORMAT_VERBOSE_-TIMESTAMPED,

NDDS::NDDS_CONFIG_LOG_PRINT_FORMAT_DEBUG,

NDDS::NDDS_CONFIG_LOG_PRINT_FORMAT_MINIMAL,

NDDS::NDDS_CONFIG_LOG_PRINT_FORMAT_MAXIMAL }

The format used to output RTI Data Distribution Service diagnostic information.

Functions

^ public delegate System::Int32 **NDDS::LogCallbackDelegate** (System::String^ msg)

A delegate for the log callback.

5.31.1 Detailed Description

Utility API's independent of the DDS standard.

5.31.2 Enumeration Type Documentation

5.31.2.1 enum NDDS::LogVerbosity

The verbosities at which RTI Data Distribution Service diagnostic information is logged.

Enumerator:

NDDS_CONFIG_LOG_VERBOSITY_SILENT No further output will be logged.

NDDS_CONFIG_LOG_VERBOSITY_ERROR Only error messages will be logged.

An error indicates something wrong in the functioning of RTI Data Distribution Service. 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 Data Distribution Service 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 Data Distribution Service objects will be logged.

NDDS_CONFIG_LOG_VERBOSITY_STATUS_REMOTE Errors, warnings, and verbose information about the lifecycles of remote RTI Data Distribution Service objects will be logged.

NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL Errors, warnings, verbose information about the lifecycles of local and remote RTI Data Distribution Service objects, and periodic information about RTI Data Distribution Service threads will be logged.

5.31.2.2 enum NDDS::LogCategory

Categories of logged messages.

The **NDDS::ConfigLogger::get_verbosity_by_category** (p. 414) and **NDDS::ConfigLogger::set_verbosity_by_category** (p. 415) can be used to specify different verbosities for different categories of messages.

Enumerator:

NDDS_CONFIG_LOG_CATEGORY_PLATFORM Log messages pertaining to the underlying platform (hardware and OS) on which RTI Data Distribution Service 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 Data Distribution Service objects are stored are in this category.

NDDS_CONFIG_LOG_CATEGORY_ENTITIES Log messages pertaining to local and remote entities and to the discovery process are in this category.

NDDS_CONFIG_LOG_CATEGORY_API Log messages pertaining to the API layer of RTI Data Distribution Service (such as method argument validation) are in this category.

5.31.2.3 enum NDDS::LogPrintFormat

The format used to output RTI Data Distribution Service diagnostic information.

Enumerator:

NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT Print message, method name, and activity context (default).

NDDS_CONFIG_LOG_PRINT_FORMAT_TIMESTAMPED Print message, method name, activity context, and timestamp.

NDDS_CONFIG_LOG_PRINT_FORMAT_VERBOSE Print message with all available context information (includes thread identifier, activity context).

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 field that may be useful for internal debug.

NDDS_CONFIG_LOG_PRINT_FORMAT_MINIMAL Print only message number and method name.

NDDS_CONFIG_LOG_PRINT_FORMAT_MAXIMAL Print all available fields.

5.31.3 Function Documentation

5.31.3.1 public delegate System::Int32 NDDS::LogCallbackDelegate (System::String^ *msg*)

A delegate for the log callback.

5.32 Unsupported Utilities

Unsupported APIs used by examples in the RTI Data Distribution Service distribution as well as in rtiddsgen-generated examples.

5.33 Durability and Persistence

APIs related to RTI Data Distribution Service Durability and Persistence. RTI Data Distribution Service offers the following mechanisms for achieving durability and persistence:

- **Durable Writer History** (p. 129)
- **Durable Reader State** (p. 129)
- **Data Durability** (p. 130)

To use any of these features, you need a relational database, which is not included with RTI Data Distribution Service. Supported databases are listed in the Release Notes.

These three features can be used separately or in combination.

5.33.1 Durable Writer History

This feature allows a **DDS::DataWriter** (p. 499) to locally persist its local history cache so that it can survive shutdowns, crashes and restarts. When an application restarts, each **DDS::DataWriter** (p. 499) 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. 499) had been temporarily disconnected from the network and then reappeared.

See also:

Configuring Durable Writer History (p. 131)

5.33.2 Durable Reader State

This feature allows a **DDS::DataReader** (p. 433) to locally persists its state and remember the data it has already received. When an application restarts, each **DDS::DataReader** (p. 433) 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. 433) before the restart will be suppressed so it is not sent over the network.

5.33.3 Data Durability

This feature is a full implementation of the OMG DDS Persistence Profile. The **DURABILITY** (p. 276) QoS lets an application configure a **DDS::DataWriter** (p. 499) such that the information written by the **DDS::DataWriter** (p. 499) survives beyond the lifetime of the **DDS::DataWriter** (p. 499). In this manner, a late-joining **DDS::DataReader** (p. 433) can subscribe and receive the information even after the **DDS::DataWriter** (p. 499) application is no longer executing. To use this feature, you need RTI Persistence Service – an optional product that can be purchased separately.

5.33.4 Durability and Persistence Based on Virtual GUID

Every modification to the global dataspace made by a **DDS::DataWriter** (p. 499) 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. 499) or **DDS::DataReader** (p. 433); 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. 499).

Several **DDS::DataWriter** (p. 499) entities can be configured with the same virtual GUID. If each of these **DDS::DataWriter** (p. 499) entities publishes a sample with sequence number '0', the sample will only be received once by the **DDS::DataReader** (p. 433) entities subscribing to the content published by the **DDS::DataWriter** (p. 499) entities.

RTI Data Distribution Service 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. 499) will be persisted in a database table with a name generated from the virtual GUID of the **DDS::DataWriter** (p. 499). If the **DDS::DataWriter** (p. 499) is restarted, it must have associated the same virtual GUID to restore its previous history.

Likewise, the state of a **DDS::DataReader** (p. 433) will be persisted in a database table whose name is generated from the **DDS::DataReader** (p. 433) virtual GUID

A **DDS::DataWriter** (p. 499)'s virtual GUID can be configured using **DDS::DataWriterProtocolQosPolicy::virtual_guid** (p. 530). Similarly, a **DDS::DataReader** (p. 433)'s virtual GUID can be configured using **DDS::DataReaderProtocolQosPolicy::virtual_guid** (p. 467).

The **DDS::PublicationBuiltinTopicData** (p. 1030) and **DDS::SubscriptionBuiltinTopicData** (p. 1233) 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. 530)
DDS::DataReaderProtocolQosPolicy::virtual_guid (p. 467).

5.33.5 Configuring Durable Writer History

To configure a **DDS::DataWriter** (p. 499) to have durable writer history, use the **PROPERTY** (p. 357) QoS policy associated with the **DDS::DataWriter** (p. 499) or the **DDS::DomainParticipant** (p. 577).

Properties defined for the **DDS::DomainParticipant** (p. 577) will be applied to all the **DDS::DataWriter** (p. 499) objects belonging to the **DDS::DomainParticipant** (p. 577), unless the property is overwritten by the **DDS::DataWriter** (p. 499).

See also:

DDS::PropertyQosPolicy (p. 1023)

The following table lists the supported durable writer history properties.

5.33.6 Configuring Durable Reader State

To configure a **DDS::DataReader** (p. 433) with durable reader state, use the **PROPERTY** (p. 357) QoS policy associated with the **DDS::DataReader** (p. 433) or **DDS::DomainParticipant** (p. 577).

A property defined in the **DDS::DomainParticipant** (p. 577) will be applicable to all the **DDS::DataReader** (p. 433) belonging to the **DDS::DomainParticipant** (p. 577) unless it is overwritten by the **DDS::DataReader** (p. 433).

See also:

DDS::PropertyQosPolicy (p. 1023)

The following table lists the supported durable reader state properties.

5.33.7 Configuring Data Durability

RTI Data Distribution Service implements DDS::DurabilityQosPolicyKind::TRANSIENT_DURABILITY_QOS and DDS::DurabilityQosPolicyKind::PERSISTENT_DURABILITY_QOS 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 online documentation.

See also:

DURABILITY (p. 276)

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.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.driver	This property tells RTI Data Distribution Service which ODBC driver to load. If the property is not specified, RTI Data Distribution Service will try to use the standard ODBC driver manager library: UnixOdbc (odbc32.dll) on UNIX/Linux systems; the Windows ODBC driver manager (libodbc.so) on Windows systems).
dds.data_writer.history.odbc_plugin.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.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.shared	If set to 1, RTI Data Distribution Service 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. 499) will create its own database connection. Default: 0 (false)
dds.data_writer.history.odbc_plugin.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 Data Distribution Service uses two caches, one for samples and one for instances. The initial and maximum sizes of these
dds.data_writer.history.odbc_plugin.instance_cache_initial_size dds.data_writer.history.odbc_plugin.instance_cache_max_size dds.data_writer.history.odbc_plugin.instance_cache_initial_samples dds.data_writer.history.odbc_plugin.instance_cache_max_samples dds.data_writer.history.odbc_plugin.instance_cache_max_samples_per_instance	These properties configure the resource limits associated with the ODBC writer history caches. To minimize the number of accesses to the database, RTI Data Distribution Service uses two caches, one for samples and one for instances. The initial and maximum sizes of these properties. The resource limits initial_instances, max_instances, initial_samples, max_samples and max_samples_per_instance in the **DDS::ResourceLimitsQosPolicy** (p. 1109) are used to configure the maximum number of samples and instances that can be stored in the

Property	Description
dds.data_reader.state.odbc.dsn	The ODBC DSN (Data Source Name) associated with the database where the **DDS::DataReader** (p. 433) 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 Data Distribution Service does not filter duplicate samples that may be coming from the same virtual writer. By default, this property is set to 1.
dds.data_reader.state.odbc.driver	This property is used to indicate which ODBC driver to load. If the property is not specified, RTI Data Distribution Service will try to use the standard ODBC driver manager library: UnixOdbc (odbc32.dll) on UNIX/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::DataReader** (p. 433) state must be restored or not once the **DDS::DataReader** (p. 433) is restarted. If this property is 0, the previous state will be deleted from the database. If it is 1, the **DDS::DataReader** (p. 433) 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 every N samples. A high frequency 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.

5.34 Configuring QoS Profiles with XML

APIs related to XML QoS Profiles.

5.34.1 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 when the following operations are called:

- **DDS::DomainParticipantFactory::create_participant** (p. 665)
- **DDS::DomainParticipantFactory::create_participant_with_profile** (p. 667)
- **DDS::DomainParticipantFactory::set_default_participant_qos_with_profile** (p. 655)
- **DDS::DomainParticipantFactory::get_default_participant_qos** (p. 656)
- **DDS::DomainParticipantFactory::set_default_library** (p. 657)
- **DDS::DomainParticipantFactory::set_default_profile** (p. 658)
- **DDS::DomainParticipantFactory::get_participant_qos_from_profile** (p. 659)
- **DDS::DomainParticipantFactory::get_topic_qos_from_profile** (p. 664)
- **DDS::DomainParticipantFactory::get_topic_qos_from_profile_w_topic_name** (p. 664)
- **DDS::DomainParticipantFactory::get_publisher_qos_from_profile** (p. 660)
- **DDS::DomainParticipantFactory::get_subscriber_qos_from_profile** (p. 661)
- **DDS::DomainParticipantFactory::get_datawriter_qos_from_profile** (p. 661)
- **DDS::DomainParticipantFactory::get_datawriter_qos_from_profile_w_topic_name** (p. 662)
- **DDS::DomainParticipantFactory::get_datareader_qos_from_profile** (p. 662)

- **DDS::DomainParticipantFactory::get_datareader_qos_from_profile_w_topic_name** (p. 663)
- **DDS::DomainParticipantFactory::get_qos_profile_libraries** (p. 665)
- **DDS::DomainParticipantFactory::get_qos_profiles** (p. 665)
- **DDS::DomainParticipantFactory::load_profiles** (p. 671)

The QoS profiles are reloaded replacing previously loaded profiles when the following operations are called:

- **DDS::DomainParticipantFactory::set_qos** (p. 669)
- **DDS::DomainParticipantFactory::reload_profiles** (p. 671)

The **DDS::DomainParticipantFactory::unload_profiles()** (p. 671) operation will free the resources associated with the XML QoS profiles.

There are five ways to configure the XML resources (listed by load order):

- The file NDDS_QOS_PROFILES.xml in $NDDSHOME/resource/qos_profiles_4.5c/xml is loaded if it exists and **DDS::ProfileQosPolicy::ignore_resource_profile** (p. 1021) in **DDS::ProfileQosPolicy** (p. 1019) is set to false (first to be loaded). An example file, `NDDS_QOS_PROFILES.example.xml`, is available for reference.
- The URL groups separated by semicolons referenced by the enviroment variable NDDS_QOS_PROFILES are loaded if they exist and **DDS::ProfileQosPolicy::ignore_environment_profile** (p. 1021) in **DDS::ProfileQosPolicy** (p. 1019) is set to false.
- The file USER_QOS_PROFILES.xml in the working directory will be loaded if it exists and **DDS::ProfileQosPolicy::ignore_user_profile** (p. 1020) in **DDS::ProfileQosPolicy** (p. 1019) is set to false.
- The URL groups referenced by **DDS::ProfileQosPolicy::url_profile** (p. 1020) in **DDS::ProfileQosPolicy** (p. 1019) will be loaded if specified.
- The sequence of XML strings referenced by **DDS::ProfileQosPolicy::string_profile** (p. 1020) will be loaded if specified (last to be loaded).

The above methods can be combined together.

5.34.2 URL

The location of the XML resources (only files and strings are supported) is specified using a URL (Uniform Resource Locator) format. For example:

File Specification: `file:///usr/local/default_dds.xml`

String Specification: str://"<dds><qos_library> . . . lt;/qos_-library>&lt/dds>"

If the URL schema name is omitted, RTI Data Distribution Service will assume a file name. For example:

File Specification: /usr/local/default_dds.xml

5.34.2.1 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 Data Distribution Service, starting from the left.

Brackets are not required for groups with a single URL.

5.34.2.2 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 enviroment variable are loaded if they exist and **DDS::ProfileQosPolicy::ignore_environment_profile** (p. 1021) is set to false

For more information on XML Configuration, refer to the User's Manual.

5.35 Publication Example

A data publication example.

5.35.1 A typical publication example

Prep

- **Create user data types using rtiddsgen** (p. 196)

Set up

- **Get the factory** (p. 140)
- **Set up participant** (p. 140)
- **Set up publisher** (p. 149)
- **Register user data type(s)** (p. 143)
- **Set up topic(s)** (p. 143)
- **Set up data writer(s)** (p. 150)

Adjust the desired quality of service (QoS)

- **Adjust QoS on entities as necessary** (p. 160)

Send data

- **Send data** (p. 151)

Tear down

- **Tear down data writer(s)** (p. 151)
- **Tear down topic(s)** (p. 144)
- **Tear down publisher** (p. 149)
- **Tear down participant** (p. 141)

5.36 Subscription Example

A data subscription example.

5.36.1 A typical subscription example

Prep

- **Create user data types using rtiddsgen** (p. 196)

Set up

- **Get the factory** (p. 140)
- **Set up participant** (p. 140)
- **Set up subscriber** (p. 152)
- **Register user data type(s)** (p. 143)
- **Set up topic(s)** (p. 143)
- **Set up data reader(s)** (p. 155)
- **Set up data reader** (p. 156) **OR Set up subscriber** (p. 152) to receive data

Adjust the desired quality of service (QoS)

- **Adjust QoS on entities as necessary** (p. 160)

Receive data

- Access received data either **via a reader** (p. 156) **OR via a subscriber** (p. 153) (possibly in a **ordered or coherent** (p. 154) manner)

Tear down

- **Tear down data reader(s)** (p. 158)
- **Tear down topic(s)** (p. 144)
- **Tear down subscriber** (p. 154)
- **Tear down participant** (p. 141)

5.37 Participant Use Cases

Working with domain partcipants. Working with domain participants.

5.37.1 Turning off auto-enable of newly created participant(s)

ˆ **Get the factory** (p. 140)

ˆ Change the value of the **ENTITY_FACTORY** (p. 304) for the **DDS::DomainParticipantFactory** (p. 649)

```
DomainParticipantFactoryQos factory_qos =
    new DomainParticipantFactoryQos();
try {
    factory.get_qos(factory_qos);
} catch (DDS.Exception) {
    Console.WriteLine(
        "***Error: failed to get domain participant factory qos");
}

// Change the QosPolicy to create disabled participants
factory_qos.entity_factory.autoenable_created_entities = false;

try {
    factory.set_qos(factory_qos);
} catch (DDS.Exception) {
    Console.WriteLine(
        "***Error: failed to set domain participant factory qos");
}
```

5.37.2 Getting the factory

ˆ Get the DomainParticipantFactory instance:

```
DomainParticipantFactory factory =
    DomainParticipantFactory.get_instance();

if (factory == null) {
    Console.WriteLine(
        "***Error: failed to get domain participant factory");
}
```

5.37.3 Setting up a participant

ˆ **Get the factory** (p. 140)

ˆ Create DomainParticipant:

```
DomainParticipantQos participant_qos = new DomainParticipantQos();
DomainParticipantListener participant_listener = null;

/* 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.
 */
String[] 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",
};

// initialize participant_qos with default values
try {
    factory.get_default_participant_qos(participant_qos);
} catch (DDS.Exception) {
    Console.WriteLine(
        "***Error: failed to get default participant qos");
}

participant_qos.discovery.initial_peers.from_array(
        NDDS_DISCOVERY_INITIAL_PEERS);

// Create the participant
DomainParticipant participant = factory.create_participant(
    domain_id,
    participant_qos,
    participant_listener,
    StatusMask.STATUS_MASK_NONE);
if (participant == null) {
    Console.WriteLine(
        "***Error: failed to create domain participant");
}
```

5.37.4 Tearing down a participant

- **Get the factory** (p. 140)

- Delete DomainParticipant:

```
try {
```

```
    factory.delete_participant(ref participant);
} catch (DDS.Exception) {
    Console.WriteLine(
        "***Error: failed to delete domain participant");
}
```

5.38 Topic Use Cases

Working with topics.

5.38.1 Registering a user data type

- **Set up participant** (p. 140)

- Register user data type of type `Foo` (p. 877) under the name "My Type"

```
type_name = "My Type";
try {
    FooTypeSupport.register_type(participant, type_name);
} catch (DDS.Exception) {
    Console.WriteLine("***Error: failed to register type");
}
```

5.38.2 Setting up a topic

- **Set up participant** (p. 140)

- **Ensure user data type is registered** (p. 143)

- Create a Topic under the name "My Topic"

```
TopicQos topic_qos = new TopicQos();
TopicListener topic_listener = null;

try {
    participant.get_default_topic_qos(topic_qos);
} catch (DDS.Exception) {
    Console.WriteLine(
        "***Error: failed to get default topic qos");
}

topic_name = "My Topic";
Topic topic = participant.create_topic(
    topic_name,
    type_name,
    topic_qos,
    topic_listener,
    StatusMask.STATUS_MASK_NONE);
if (topic == null) {
    Console.WriteLine("***Error: failed to create topic");
}
```

5.38.3 Tearing down a topic

ˆ Delete Topic:

```
try {
    participant.delete_topic(ref topic);
} catch (DDS.Exception) {
    Console.WriteLine("***Error: failed to delete topic");
}
```

5.39 FlowController Use Cases

Working with flow controllers.

5.39.1 Creating a flow controller

- **Set up participant** (p. 140)

- Create a flow controller

```
DDS.FlowController controller = null;
DDS.FlowControllerProperty_t property = new DDS.FlowControllerProperty_t();

try {
   participant.get_default_flowcontroller_property(property);
} catch (DDS.Exception) {
    System.Console.WriteLine(
        "***Error: failed to get default flow controller property");
}

// optionally modify flow controller property values

controller = participant.create_flowcontroller(
    "my flow controller name", property);

if (controller == null) {
    System.Console.WriteLine(
        "***Error: failed to create flow controller");
}
```

5.39.2 Flow controlling a data writer

- **Set up participant** (p. 140)

- **Create flow controller** (p. 145)

- Create an asynchronous data writer, **FooDataWriter** (p. 879), of user data type `Foo` (p. 877):

```
DataWriterQos writer_qos = new DataWriterQos();

// MyWriterListener is user defined and
// extends DataWriterListener
MyWriterListener writer_listener = new MyWriterListener(); // or = null

try {
    publisher.get_default_datawriter_qos(writer_qos);
} catch (DDS.Exception) {
    // ... error
```

```
}

// Change the writer QoS to publish asnychronously
writer_qos.publish_mode.kind =
    PublishModeQosPolicyKind.ASYNCHRONOUS_PUBLISH_MODE_QOS;

// Setup to use the previously created flow controller
writer_qos.publish_mode.flow_controller_name =
    "my flow controller name";

// Samples queued for asynchronous write are subject to the History Qos policy
writer_qos.history.kind = HistoryQosPolicyKind.KEEP_ALL_HISTORY_QOS;

FooDataWriter writer = (FooDataWriter) publisher.create_datawriter(
                                                topic,
                                                writer_qos,
                                                writer_listener,
                            STATUS_MASK_ALL);
if (writer == null) {
    // ... error
}

// Send data asynchronously...

// Wait for asynchronous send completes, if desired
try {
    writer.wait_for_asynchronous_publishing(timout);
} catch (DDS.Exception) {
    System.Console.WriteLine(
        "***Error: failed to wait for asynchronous publishing");
}
```

5.39.3 Using the built-in flow controllers

RTI Data Distribution Service provides several built-in flow controllers.

The DDS::DEFAULT_FLOW_CONTROLLER_NAME built-in flow controller provides the basic asynchronous writer behavior. When calling **DDS::TypedDataWriter::write** (p. 1376), the call signals the **DDS::Publisher** (p. 1044) asynchronous publishing thread (**DDS::PublisherQos::asynchronous_publisher** (p. 1075)) to send the actual data. As with any DDS::PublishModeQosPolicyKind::ASYNCHRONOUS_PUBLISH_MODE_QOS **DDS::DataWriter** (p. 499), the **DDS::TypedDataWriter::write** (p. 1376) call returns immediately afterwards. The data is sent immediately in the context of the **DDS::Publisher** (p. 1044) asynchronous publishing thread.

When using the DDS::FIXED_RATE_FLOW_CONTROLLER_NAME flow controller, data is also sent in the context of the **DDS::Publisher** (p. 1044) asynchronous publishing thread, but at a regular fixed interval. The thread accumulates samples from different **DDS::DataWriter** (p. 499) instances and generates data on the wire only once per

DDS::FlowControllerTokenBucketProperty_t::period (p. 875).

In contrast, the DDS::ON_DEMAND_FLOW_CONTROLLER_NAME flow controller permits flow only when **DDS::FlowController::trigger_flow** (p. 869) is called. The data is still sent in the context of the **DDS::Publisher** (p. 1044) asynchronous publishing thread. The thread accumulates samples from different **DDS::DataWriter** (p. 499) instances (across any **DDS::Publisher** (p. 1044)) and sends all data since the previous trigger.

The properties of the built-in **DDS::FlowController** (p. 867) instances can be adjusted.

- **Set up participant** (p. 140)
- Lookup built-in flow controller

```
FlowController controller = null;

controller = participant.lookup_flowcontroller(
    FlowController.DEFAULT_FLOW_CONTROLLER_NAME);

// This should never happen, built-in flow controllers are always created
if (controller == null) {
    System.Console.WriteLine("***Error: failed to lookup flow controller");
}
```

- Change property of built-in flow controller, if desired

```
DDS.FlowControllerProperty_t property = new DDS.FlowControllerProperty_t();

// Get the property of the flow controller
try {
    controller.get_property(property);
} catch (DDS.Exception) {
    System.Console.WriteLine(
        "***Error: failed to get flow controller property");
}

// Change the property value as desired
property.token_bucket.period.sec = 2;
property.token_bucket.period.nanosec = 0;

// Update the flow controller property
try {
   controller.set_property(property);
} catch (DDS.Exception) {
    System.Console.WriteLine(
        "***Error: failed to set flow controller property");
}
```

- **Create a data writer using the correct flow controller name** (p. 145)

5.39.4 Shaping the network traffic for a particular transport

- **Set up participant** (p. 140)
- **Create the transports** (p. 165)
- **Create a separate flow controller for each transport** (p. 145)
- Configure **DDS::DataWriter** (p. 499) instances to only use a single transport
- **Associate all data writers using the same transport to the corresponding flow controller** (p. 145)
- For each transport, the corresponding flow controller limits the network traffic based on the token bucket properties

5.39.5 Coalescing multiple samples in a single network packet

- **Set up participant** (p. 140)
- **Create a flow controller with a desired token bucket period** (p. 145)
- **Associate the data writer with the flow controller** (p. 145)
- 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).

5.40 Publisher Use Cases

Working with publishers.

5.40.1 Setting up a publisher

- **Set up participant** (p. 140)

- Create a Publisher

```
PublisherQos publisher_qos = new PublisherQos();
PublisherListener publisher_listener = null;

try {
    participant.get_default_publisher_qos(publisher_qos);
} catch (DDS.Exception) {
    Console.WriteLine("***Error: failed to get default publisher qos");
}

Publisher publisher = participant.create_publisher(
    publisher_qos,
    publisher_listener,
    StatusMask.STATUS_MASK_NONE);
if (publisher == null) {
    Console.WriteLine("***Error: failed to create publisher");
}
```

5.40.2 Tearing down a publisher

- Delete Publisher:

```
try {
    participant.delete_publisher(ref publisher);
} catch (DDS.Exception) {
    Console.WriteLine("***Error: failed to delete publisher");
}
```

5.41 DataWriter Use Cases

Working with data writers.

5.41.1 Setting up a data writer

- **Set up publisher** (p. 149)
- **Set up a topic** (p. 143)
- Create a data writer, **FooDataWriter** (p. 879), of user data type `Foo` (p. 877):

```
DataWriterQos writer_qos = new DataWriterQos();
DataWriterListener writer_listener = null;

try {
    publisher.get_default_datawriter_qos(writer_qos);
} catch (DDS.Exception) {
    Console.WriteLine(
        "***Error: failed to get default datawriter qos");
}

DataWriter writer = publisher.create_datawriter(topic,
    writer_qos,
    writer_listener,
    StatusMask.STATUS_MASK_NONE);
if (writer == null) {
    Console.WriteLine("***Error: failed to create writer");
}
```

5.41.2 Managing instances

- Getting an instance "key" value of user data type `Foo` (p. 877)

```
Foo data = ...;     // user data

try {
    writer.get_key_value(data, ref instance_handle);
} catch (DDS.Exception) {
    // ... check for cause of failure
}
```

- Registering an instance of type `Foo` (p. 877)

```
InstanceHandle_t instance_handle = InstanceHandle_t.HANDLE_NIL;

instance_handle = writer.register_instance(data);
```

- Unregistering an instance of type `Foo` (p. 877)

```
try {
    writer.unregister_instance(data, ref instance_handle);
} catch (DDS.Exception) {
    // ... check for cause of failure
}
```

- Disposing of an instance of type `Foo` (p. 877)

```
try {
    writer.dispose(data, ref instance_handle);
} catch (DDS.Exception) {
    // ... check for cause of failure
}
```

5.41.3 Sending data

- **Set up data writer** (p. 150)

- **Register instance** (p. 150)

- Write instance of type `Foo` (p. 877)

```
Foo data = new Foo();     // user data
InstanceHandle_t instance_handle =
    InstanceHandle_t.HANDLE_NIL; // or a valid registered handle

try {
    writer.write(data, ref instance_handle);
} catch (DDS.Exception) {
    Console.WriteLine("***Error: failed to write data");
}
```

5.41.4 Tearing down a data writer

- Delete DataWriter:

```
try {
    publisher.delete_datawriter(ref writer);
} catch (DDS.Exception) {
    Console.WriteLine("***Error: failed to delete writer");
}
```

5.42 Subscriber Use Cases

Working with subscribers.

5.42.1 Setting up a subscriber

- **Set up participant** (p. 140)

- Create a Subscriber

```
SubscriberQos subscriber_qos = new SubscriberQos();
SubscriberListener subscriber_listener = null;

try {
    participant.get_default_subscriber_qos(subscriber_qos);
} catch (DDS.Exception) {
    Console.WriteLine(
        "***Error: failed to get default subscriber qos");
}

Subscriber subscriber = participant.create_subscriber(
    subscriber_qos,
    subscriber_listener,
    StatusMask.STATUS_MASK_NONE);
if (subscriber == null) {
    Console.WriteLine(
        "***Error: failed to create subscriber");
}
```

5.42.2 Set up subscriber to access received data

- **Set up subscriber** (p. 152)

- Set up to handle the DATA_ON_READERS_STATUS status, in one or both of the following two ways.

- **Enable DATA_ON_READERS_STATUS for the SubscriberListener associated with the subscriber** (p. 161)
  - The processing to handle the status change is done in the SubscriberListener.on_data_on_readers() method of the attached listener.
  - Typical processing will **access the received data** (p. 153), either in arbitrary order or in a **coherent and ordered manner** (p. 154).

- **Enable DATA_ON_READERS_STATUS for the StatusCondition associated with the subscriber** (p. 162)

- The processing to handle the status change is done **when the subscriber's attached status condition is triggered** (p. 163) and the DATA_ON_READERS_STATUS status on the subscriber is changed.
- Typical processing will **access the received data** (p. 153), either in an arbitrary order or in a **coherent and ordered manner** (p. 154).

5.42.3 Access received data via a subscriber

^ **Ensure subscriber is set up to access received data** (p. 152)

^ Get the list of readers that have data samples available:

```
// holder for list/set of readers
DataReaderSeq reader_seq = new DataReaderSeq();
int max_samples = ResourceLimitsQosPolicy.LENGTH_UNLIMITED;
SampleStateKind sample_state_mask =
    SampleStateKind.ANY_SAMPLE_STATE;
ViewStateKind view_state_mask = ViewStateKind.ANY_VIEW_STATE;
InstanceStateKind instance_state_mask =
    InstanceStateKind.ANY_INSTANCE_STATE;

try {
    subscriber.get_datareaders(
        reader_seq,
            sample_state_mask,
            view_state_mask,
            instance_state_mask);
} catch (DDS.Exception) {
    Console.WriteLine(
        "***Error: failed to access received data via subscriber");
    return;
}
```

^ Upon successfully getting the list of readers with data, process the data readers to either:

- **Read the data in each reader** (p. 157), **OR**
- **Take the data in each reader** (p. 156)

If the intent is to access the data **coherently or in order** (p. 154), the list of data readers *must* be processed in the order returned:

```
for (int i = 0; i < reader_seq.length; ++i) {
    DataReader reader = reader_seq.get_at(i);

  // Take the data from reader,
  //            OR
  // Read the data from reader
}
```

- **Alternatively**, call Subscriber.notify_datareaders() to invoke the DataReaderListener for each of the data readers.

```
try {
        subscriber.notify_datareaders();
} catch (DDS.Exception) {
    Console.WriteLine(
        "***Error: failed to notify datareaders");
}
```

5.42.4 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 PresentationQosPolicy attached to a Subscriber:

- **Ensure subscriber is set up to access received data** (p. 152)

- Indicate that data will be accessed via the subscriber:

```
try {
    subscriber.begin_access();
} catch (DDS.Exception) {
    // ... check for cause of failure
}
```

- **Access received data via the subscriber, making sure that the data readers are processed in the order returned.** (p. 153)

- Indicate that the data access via the subscriber is done:

```
try {
    subscriber.end_access();
} catch (DDS.Exception) {
    // ... check for cause of failure
}
```

5.42.5 Tearing down a subscriber

- Delete Subscriber:

```
try {
    participant.delete_subscriber(ref subscriber);
} catch (DDS.Exception) {
    Console.WriteLine(
        "***Error: failed to delete subscriber");
}
```

5.43 DataReader Use Cases

Working with data readers.

5.43.1 Setting up a data reader

- **Set up subscriber** (p. 152)
- **Set up a topic** (p. 143)
- Create a data reader, **FooDataReader** (p. 878), of user data type `Foo` (p. 877):

```
DataReaderQos reader_qos = new DataReaderQos();
DataReaderListener reader_listener = null;

try {
    subscriber.get_default_datareader_qos(reader_qos);
} catch (DDS.Exception) {
    Console.WriteLine(
        "***Error: failed to get default datareader qos");
}

DataReader reader = subscriber.create_datareader(
    topic,
    reader_qos,
    reader_listener,
    StatusMask.STATUS_MASK_NONE);
if (reader == null) {
        Console.WriteLine("***Error: failed to create reader");
}
```

5.43.2 Managing instances

- Given a data reader

```
FooDataReader reader = ...;
```

- Getting an instance "key" value of user data type `Foo` (p. 877)

```
Foo data = new Foo(); // user data of type Foo
// ...
try {
    reader.get_key_value(data, instance_handle);
} catch (DDS.Exception) {
    // ... check for cause of failure
}
```

5.43.3 Set up reader to access received data

- **Set up data reader** (p. 155)

- Set up to handle the DATA_AVAILABLE_STATUS status, in one or both of the following two ways.

- **Enable DATA_AVAILABLE_STATUS for the DataReaderListener associated with the data reader** (p. 161)
  - The processing to handle the status change is done in the **DDS::DataReaderListener::on_data_available** (p. 463) method of the attached listener.
  - Typical processing will **access the received data** (p. 156).

- **Enable DATA_AVAILABLE_STATUS for the StatusCondition associated with the data reader** (p. 162)
  - The processing to handle the status change is done **when the data reader's attached status condition is triggered** (p. 163) and the DATA_AVAILABLE_STATUS status on the data reader is changed.
  - Typical processing will **access the received data** (p. 156).

5.43.4 Access received data via a reader

- **Ensure reader is set up to access received data** (p. 156)

- Access the received data, by either:
  - **Taking the received data in the reader** (p. 156), **OR**
  - **Reading the received data in the reader** (p. 157)

5.43.5 Taking data

- **Ensure reader is set up to access received data** (p. 156)

- Take samples of user data type `T`. The samples are removed from the Service. The caller is responsible for deallocating the buffers.

```
FooSeq data_seq = new FooSeq(); // sequence of user data type Foo
SampleInfoSeq info_seq = new SampleInfoSeq(); // sequence of SampleInfo
int max_samples = ResourceLimitsQosPolicy.LENGTH_UNLIMITED;
SampleStateKind sample_state_mask =
    SampleStateKind.ANY_SAMPLE_STATE;
```

```
ViewStateKind view_state_mask =
    ViewStateKind.ANY_VIEW_STATE;
InstanceStateKind instance_state_mask =
    InstanceStateKind.ANY_INSTANCE_STATE;

try {
    reader.take(data_seq, info_seq,
        max_samples,
        sample_state_mask,
        view_state_mask,
        instance_state_mask);
} catch (Retcode_NoData) {
    return;
} catch (DDS.Exception) {
    Console.WriteLine(
        "***Error: failed to access data from the reader");
}
```

ˆ Use the received data

```
// Use the received data samples 'data_seq' and associated
// meta-information 'info_seq'
for (int i = 0; i < data_seq.length; ++i) {
    SampleInfo info = info_seq.get_at(i);
    if (!info.valid_data) {
        continue;
    }
    Foo data = data_seq.get_at(i);

    Console.WriteLine(
        "  Data = (%d, %d)   sample_state = %d",
        data.x, data.y, info.sample_state);
}
```

ˆ 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.

```
try {
    reader.return_loan(data_seq, info_seq);
} catch (DDS.Exception) {
    Console.WriteLine("***Error: failed to return loan");
}
```

5.43.6 Reading data

ˆ **Ensure reader is set up to access received data** (p. 156)

ˆ Read samples of user data type `Foo` (p. 877). The samples are not removed from the Service. It remains responsible for deallocating the buffers.

```
FooSeq data_seq = new FooSeq(); // sequence of user data type Foo
SampleInfoSeq info_seq = new SampleInfoSeq(); // sequence of SampleInfo
int max_samples = ResourceLimitsQosPolicy.LENGTH_UNLIMITED;
SampleStateKind sample_state_mask =
    SampleStateKind.ANY_SAMPLE_STATE;
ViewStateKind view_state_mask =
    ViewStateKind.ANY_VIEW_STATE;
InstanceStateKind instance_state_mask =
    InstanceStateKind.ANY_INSTANCE_STATE;

try {
    reader.read(data_seq, info_seq,
        max_samples,
        sample_state_mask,
        view_state_mask,
        instance_state_mask);
} catch (Retcode_NoData) {
    return;
} catch (DDS.Exception) {
    Console.WriteLine(
        "***Error: failed to access data from the reader");
}
```

ˆ Use the received data

```
// Use the received data samples 'data_seq' and associated
// meta-information 'info_seq'
for (int i = 0; i < data_seq.length; ++i) {
    SampleInfo info = info_seq.get_at(i);
    if (!info.valid_data) {
        continue;
    }
    Foo data = data_seq.get_at(i);

    Console.WriteLine(
        "  Data = (%d, %d)   sample_state = %d",
        data.x, data.y, info.sample_state);
}
```

ˆ Return the data samples and the information buffers back to the middleware

```
try {
    reader.return_loan(data_seq, info_seq);
} catch (DDS.Exception) {
    Console.WriteLine("***Error: failed to return loan");
}
```

5.43.7 Tearing down a data reader

ˆ Delete DataReader:

```
try {
    subscriber.delete_datareader(ref reader);
} catch (DDS.Exception) {
    Console.WriteLine("***Error: failed to delete reader");
}
```

5.44 Entity Use Cases

Working with entities.

5.44.1 Enabling an entity

- To enable an **DDS::Entity** (p. 845)

```
try {
    entity.enable();
} catch (DDS.Exception) {
    Console.WriteLine("***Error: failed to enable entity");
}
```

5.44.2 Checking if a status changed on an entity.

- Given an **DDS::Entity** (p. 845) and a DDS::StatusKind to check for, get the list of statuses that have changed since the last time they were respectively cleared.

```
StatusMask status_changes_mask = entity.get_status_changes();
```

- Check if `status_kind` was changed since the last time it was cleared. A plain communication status change is cleared when the status is read using the entity's `get_<plain communication status>()` method. A read communication status change is cleared when the data is taken from the middleware via a TDataReader_take() call [see **Changes in Status** (p. 240) for details].

```
if (((int) status_changes_mask & (int) status_kind) != 0) {
    return true;
} else {           /* ... YES, status_kind changed ... */
    return false;  /* ... NO, status_kind did NOT change ... */
}
```

5.44.3 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. 847)

```
try {
    entity.get_qos(qos);
} catch (DDS.Exception) {
    Console.WriteLine("***Error: failed to get qos");
}
```

- Change the desired qos policy fields

```
// Change the desired qos policies
// qos.policy.field = ...
```

- Set the qos using **set_qos (abstract)** (p. 846).

```
try {
    entity.set_qos(qos);
} catch (Retcode_ImmutablePolicy) {
    Console.WriteLine(
        "***Error: tried changing a policy that can only be" +
        "          set at entity creation time");
} catch (Retcode_InconsistentPolicy) {
    Console.WriteLine(
        "***Error: tried changing a policy to a value inconsistent" +
        "          with other policy settings");
} catch (DDS.Exception) {
    Console.WriteLine("***Error: some other failure");
}
```

5.44.4 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 <Entity>Listener ...
public class MyEntityListener : Listener {
    // ... methods defined by <Entity>Listener ...
};
```

- Get an entity's listener using **get_listener (abstract)** (p. 848)

```
entity_listener = entity.get_listener();
```

- Enable `status_kind` for the `listener`

```
enabled_status_list |= status_kind;
```

- Disable `status_kind` for the `listener`

```
enabled_status_list &= ~status_kind;
```

- Set an entity's listener to `entity_listener` using **set_listener (abstract)** (p. 847). Only enable the listener for the statuses specified by the `enabled_status_list`.

```
try {
    entity.set_listener(entity_listener, enabled_status_list);
} catch (DDS.Exception) {
    Console.WriteLine("***Error: setting entity listener");
}
```

5.44.5 Enabling/Disabling statuses associated with a status condition

Upon entity creation, by default, all the statuses are *enabled* for the DDS_StatusCondition associated with the entity.

Once an entity has been created, the list of statuses for which the DDS_StatusCondition is triggered can be manipulated as follows.

- Given an `entity`, a `status_kind`, and the associated `status_condition`:

```
statuscondition = entity.get_statuscondition();
```

- Get the list of statuses enabled for the `status_condition`

```
enabled_status_list = statuscondition.get_enabled_statuses();
```

- Check if the given `status_kind` is enabled for the `status_condition`

```
if (((int) enabled_status_list & (int) status_kind) != 0) {
    //... YES, status_kind is enabled ...
} else {
    // ... NO, status_kind is NOT enabled ...
}
```

- Enable `status_kind` for the `status_condition`

```
try {
    statuscondition.set_enabled_statuses(
        (StatusMask) ((int) enabled_status_list | (int) status_kind));
} catch (DDS.Exception) {
    /* ... check for cause of failure */
}
```

- Disable `status_kind` for the `status_condition`

```
try {
    statuscondition.set_enabled_statuses(
        (StatusMask) ((int) enabled_status_list & ~(int) status_kind));
} catch (DDS.Exception) {
    /* ... check for cause of failure */
}
```

5.45 Waitset Use Cases

Using wait-sets and conditions.

5.45.1 Setting up a wait-set

ˆ Create a wait-set

```
WaitSet waitset = new WaitSet();
```

ˆ Attach conditions

```
Condition cond1 = entity.get_statuscondition();
Condition cond2 = reader.create_readcondition(
SampleStateKind.NOT_READ_SAMPLE_STATE,
    ViewStateKind.ANY_VIEW_STATE,
    InstanceStateKind.ANY_INSTANCE_STATE);
Condition cond3 = new GuardCondition();

try {
    waitset.attach_condition(cond1);
    waitset.attach_condition(cond2);
    waitset.attach_condition(cond3);
} catch (DDS.Exception) {
    // ... error
}
```

5.45.2 Waiting for condition(s) to trigger

ˆ **Set up a wait-set** (p. 163)

ˆ Wait for a condition to trigger or `timeout`, whichever occurs first

```
Duration_t timeout;
timeout.sec = 0;
timeout.nanosec = 1000000; // 1ms

// holder for active conditions
ConditionSeq active_conditions = new ConditionSeq();

bool is_cond1_triggered = false;
bool is_cond2_triggered = false;

try {
    waitset.wait(active_conditions, timeout);

    // check if "cond1" or "cond2" are triggered:
    for (int i = 0; i < active_conditions.length; ++i) {
        if (active_conditions.get_at(i) == cond1) {
             Console.WriteLine("Cond1 was triggered!");
```

```
                is_cond1_triggered = true;
            }
            if (active_conditions.get_at(i) == cond2) {
                Console.WriteLine("Cond2 was triggered!");
                is_cond2_triggered = true;
            }
        }

        if (is_cond1_triggered) {
            // ... do something because "cond1" was triggered ...
        }

        if (is_cond2_triggered) {
            // ... do something because "cond2" was triggered ...
        }
    } catch (DDS.Retcode_Timeout) {
        // timeout!
        Console.WriteLine(
            "Wait timed out!! None of the conditions was triggered.");
    } catch (DDS.Exception) {
        // ... check for cause of failure
        throw;
    }
```

5.45.3 Tearing down a wait-set

ˆ Delete the wait-set

```
waitset.Dispose();
waitset = null;
```

5.46 Transport Use Cases

Working with pluggable transports.

5.46.1 Changing the automatically registered built-in transports

- The DDS::TransportBuiltinKindMask::TRANSPORTBUILTIN_MASK_DEFAULT specifies the transport plugins that will be automatically registered with a newly created **DDS::DomainParticipant** (p. 577) by default.

- This default can be changed by changing the value of the value of **TRANSPORT_BUILTIN** (p. 321) Qos Policy on the **DDS::DomainParticipant** (p. 577)

- To change the **DDS::DomainParticipantQos::transport_builtin** (p. 685) Qos Policy:

```
DomainParticipantQos participant_qos = new DomainParticipantQos();

factory.get_default_participant_qos(participant_qos);

participant_qos.transport_builtin.mask =
    (int) TransportBuiltinKind.TRANSPORTBUILTIN_SHMEM |
    (int) TransportBuiltinKind.TRANSPORTBUILTIN_UDPv4;
```

5.47 Filter Use Cases

Working with data filters.

5.47.1 Introduction

RTI Data Distribution Service supports filtering data either during the exchange from **DDS::DataWriter** (p. 499) to **DDS::DataReader** (p. 433), or after the data has been stored at the **DDS::DataReader** (p. 433).

Filtering during the exchange process is performed by a **DDS::ContentFilteredTopic** (p. 419), which is created by the **DDS::DataReader** (p. 433) 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. 433) is performed by creating a **DDS::QueryCondition** (p. 1082), which can then used to check for matching samples, be alerted when matching samples arrive, or retrieve matching samples through use of the **DDS::TypedDataReader::read_w_condition** (p. 1349) or **DDS::TypedDataReader::take_w_condition** (p. 1350) functions. (Conditions may also be used with the APIs **DDS::TypedDataReader::read_next_instance_w_condition** (p. 1361) and **DDS::TypedDataReader::take_next_instance_w_condition** (p. 1362).)

Filtering may be performed on any topic, either keyed or un-keyed, except the **Built-in Topics** (p. 42). Filtering may be perfomed on any field, subset of fields, or combination of fields, subject only to the limitations of the filter syntax, and some restrictions against filtering some *sparse value types* of the **Dynamic Data** (p. 73) API.

RTI Data Distribution Service contains built in support for filtering using SQL syntax, described in the **Queries and Filters Syntax** (p. 184) module.

5.47.1.1 Overview of ContentFilteredTopic

Each **DDS::ContentFilteredTopic** (p. 419) is created based on an existing **DDS::Topic** (p. 1258). The **DDS::Topic** (p. 1258) specifies the **field_names** and **field_types** of the data contained within the topic. The **DDS::ContentFilteredTopic** (p. 419), by means of its `filter_expression` and `expression_parameters`, futher specifies the *values* of the data which the **DDS::DataReader** (p. 433) wishes to receive.

Once the **DDS::ContentFilteredTopic** (p. 419) has been created, a **DDS::DataReader** (p. 433) can be created using the filtered topic. The filter's characteristics are exchanged between the **DDS::DataReader** (p. 433) and any matching **DDS::DataWriter** (p. 499) during the discovery processs.

If the **DDS::DataWriter** (p. 499) allows (by **DDS::DataWriterResourceLimitsQosPolicy::max_remote_reader_filters** (p. 554)) and the number of filtered **DDS::DataReader** (p. 433) is less than or equal to 32, and the **DDS::DataReader** (p. 433) 's **DDS::TransportMulticastQosPolicy** (p. 1287) is empty, then the **DDS::DataWriter** (p. 499) will performing filtering and send to the **DDS::DataReader** (p. 433) only those samples that meet the filtering criteria.

If disallowed by the **DDS::DataWriter** (p. 499), or if more than 32 **DDS::DataReader** (p. 433) require filtering, or the **DDS::DataReader** (p. 433) has set the **DDS::TransportMulticastQosPolicy** (p. 1287), then the **DDS::DataWriter** (p. 499) sends all samples to the **DDS::DataReader** (p. 433), and the **DDS::DataReader** (p. 433) discards any samples that do not meet the filtering criteria.

Although the `filter_expression` cannot be changed once the **DDS::ContentFilteredTopic** (p. 419) has been created, the `expression_parameters` can be modified using **DDS::ContentFilteredTopic::set_expression_parameters** (p. 422). Any changes made to the filtering criteria by means of **DDS::ContentFilteredTopic::set_expression_parameters** (p. 422), will be conveyed to any connected **DDS::DataWriter** (p. 499). 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. 433) connects to a **DDS::DataWriter** (p. 499) that *re-sends* its data, the re-sent samples will be subjected to the new filtering criteria.

5.47.1.2 Overview of QueryCondition

DDS::QueryCondition (p. 1082) combine aspects of the content filtering capabilities of **DDS::ContentFilteredTopic** (p. 419) with state filtering capabilities of **DDS::ReadCondition** (p. 1084) to create a reconfigurable means of filtering or searching data in the **DDS::DataReader** (p. 433) queue.

DDS::QueryCondition (p. 1082) may be created on a disabled **DDS::DataReader** (p. 433), or after the **DDS::DataReader** (p. 433) has been enabled. If the **DDS::DataReader** (p. 433) is enabled, and has already recevied and stored samples in its queue, then all data samples in the are filtered against the **DDS::QueryCondition** (p. 1082) filter criteria at the time that the **DDS::QueryCondition** (p. 1082) is created. (Note that an exclusive lock is held on the **DDS::DataReader** (p. 433) sample queue for the duration of the **DDS::QueryCondition** (p. 1082) creation).

Once created, incoming samples are filtered against all **DDS::QueryCondition** (p. 1082) filter criteria at the time of their arrival and storage into the **DDS::DataReader** (p. 433) queue.

The number of **DDS::QueryCondition** (p. 1082) filters that an individual **DDS::DataReader** (p. 433) may create is set by **DDS::DataReaderResourceLimitsQosPolicy::max_query_condition_filters** (p. 495), to an upper maximum of 32.

5.47.2 Filtering with ContentFilteredTopic

- **Set up subscriber** (p. 152)
- **Set up a topic** (p. 143)
- Create a ContentFilteredTopic, of user data type `Foo` (p. 877):

```
String cft_param_list[] = {"1", "100"};
StringSeq cft_parameters = new StringSeq(java.util.Arrays.asList(cft_param_list));

ContentFilteredTopic cft = participant.create_contentfilteredtopic(
                                "ContentFilteredTopic",
                                topic,
                                "value > %0 AND value < %1",
                                cft_parameters);
}
if (cft == null) {
    System.err.println("create_contentfilteredtopic error\n");
    return;
}
```

- Create a FooReader using the ContentFilteredTopic:

```
FooDataReader reader = (FooDataReader)
                           subscriber.create_datareader(
                               cft,
                               datareader_qos, // or Subscriber.DATAREADER_QOS_DEFAULT //
                               listener, // or null //
                               StatusKind.STATUS_MASK_ALL);

if (reader == null) {
    System.err.println("create_datareader error\n");
    return;
}
```

  Once setup, reading samples with a **DDS::ContentFilteredTopic** (p. 419) is exactly the same as normal reads or takes, as decribed in **DataReader Use Cases** (p. 155).

- Changing filter crieria using set_expression_parameters:

```
cft_parameters.set(0, "5");
cft_parameters.set(1, "9");
cft.set_expression_parameters(cft_parameters);
```

5.47.3 Filtering with Query Conditions

- Given a data reader of type **Foo** (p. 877)

```
FooDataReader reader = ...;
```

- Creating a QueryCondition

```
QueryCondition queryCondition = null;
String qc_param_list[] = {"1", "100"};

StringSeq qc_parameters = new StringSeq(java.util.Arrays.asList(cft_param_list));
queryCondition = reader.create_querycondition(SampleStateKind.NOT_READ_SAMPLE_STATE,
                                              ViewStateKind.ANY_VIEW_STATE,
                                              InstanceStateKind.ANY_INSTANCE_STATE,
                                              "value > %0 AND value < %1",
                                              qc_parameters);
if (queryCondition == null) {
    System.err.println("create_querycondition error\n");
    return;
}
```

- Reading matching samples with a **DDS::QueryCondition** (p. 1082)

```
FooSeq _dataSeq = new FooSeq();
SampleInfoSeq _infoSeq = new SampleInfoSeq();

try {
    reader.read_w_condition(_dataSeq, _infoSeq,
                            ResourceLimitsQosPolicy.LENGTH_UNLIMITED,
                            queryCondition);
    for(int i = 0; i < _dataSeq.size(); ++i) {
        SampleInfo info = (SampleInfo)_infoSeq.get(i);
        if (info.valid_data) {
            // --- Process data here --- //
        }
    }
} catch (RETCODE_NO_DATA noData) {
    // No data to process
} finally {
    reader.return_loan(_dataSeq, _infoSeq);
}
```

- **DDS::QueryCondition::set_query_parameters** (p. 1083) is used similarly to **DDS::ContentFilteredTopic::set_expression_parameters** (p. 422), and the same coding techniques can be used.

- Any **DDS::QueryCondition** (p. 1082) that have been created must be deleted before the **DDS::DataReader** (p. 433) can be deleted. This can be done using **DDS::DataReader::delete_contained_entities** (p. 441) or manually as in:

```
retcode = reader.delete_readcondition(queryCondition);
```

5.47.4 Filtering Performance

Although RTI Data Distribution Service 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 field only filters, the **DDS::DataReader** (p. 433) 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. 433), the filter results are retrieved from 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. 433), for either **DDS::ContentFilteredTopic** (p. 419) or **DDS::QueryCondition** (p. 1082). This does *not* apply to filtering perfomed for **DDS::ContentFilteredTopic** (p. 419) by the **DDS::DataWriter** (p. 499).

5.48 Large Data Use Cases

Working with large data types.

5.48.1 Introduction

RTI Data Distribution Service supports data types whose size exceeds the maximum message size of the underlying transports. A **DDS::DataWriter** (p. 499) 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. 433) 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. 1112) and DDS::HistoryQosPolicyKind::KEEP_LAST_HISTORY_QOS.

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.

5.48.2 Writing Large Data

In order to use the large data feature with the DDS::ReliabilityQosPolicyKind::RELIABLE_RELIABILITY_QOS setting, the **DDS::DataWriter** (p. 499) must be configured as an asynchronous writer (DDS::PublishModeQosPolicyKind::ASYNCHRONOUS_PUBLISH_MODE_QOS) with associated **DDS::FlowController** (p. 867).

While the use of an asynchronous writer and flow controller is optional when using the DDS::ReliabilityQosPolicyKind::BEST_EFFORT_RELIABILITY_QOS 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. 150)
- **Add flow control** (p. 145)

5.48.3 Receiving Large Data

Large data is supported by default and in most cases, no further changes are required.

The **DDS::DataReaderResourceLimitsQosPolicy** (p. 486) allows tuning the resources available to the **DDS::DataReader** (p. 433) for reassembling

fragmented large data.

- ^ **Set up reader** (p. 155)

5.49 Documentation Roadmap

This section contains a roadmap for the new user with pointers on what to read first.

If you are new to RTI Data Distribution Service, 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.

- If you are a .NET user, you may want to read the **.Net Language Support** (p. 177) section of this online documentation before moving on to the programming interfaces themselves. .Net is language-neutral; this section explains how to use the APIs described here from various .Net-compatible languages.

- The documentation in the **DDS API Reference** (p. 179) 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. 138) and **Subscription Example** (p. 139) 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** (p. 196).

- To integrate similar code into your own application and build system, you will likely need to refer to the `Platform Notes`.

5.50 Conventions

This section describes the conventions used in the API documentation.

5.50.1 Unsupported Features

[**Not supported (optional)**] This note means that the optional feature from the DDS specification is not supported in the current release.

5.50.2 API Naming Conventions

5.50.2.1 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. 845) subtypes, the **DDS::Condition** (p. 408) subtypes, and **DDS::WaitSet** (p. 1411). 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.

5.50.3 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::Exception** (p. 861) do not show up in the compound list or indices. This is a known limitation in the generated HTML.

5.50.4 Stereotypes

Commonly used stereotypes in the API documentation include the following.

5.50.4.1 Extensions

- ˆ *<<**eXtension**>> (p. 174)*
  - An RTI Data Distribution Service 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.

5.50.4.2 Types

ˆ *<<**interface**>>* *(p. 175)*

- Pure interface type with *no state*.
- Languages such as Java natively support the concept of an *interface* type, which is a collection of method signatures devoid of any dynamic state.
- In C++, this is achieved via a class with all *pure virtual* methods and devoid of any instance variables (ie no dynamic state).
- Interfaces are generally organized into a type hierarchy. Static typecasting along the interface type hierarchy is "safe" for valid objects.

ˆ *<<**generic**>>* *(p. 175)*

- 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. 880), FooDataType, **FooDataWriter** (p. 879), and **FooDataReader** (p. 878).
- 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.

ˆ *<<**singleton**>>* *(p. 175)*

- Singleton class. There is a single instance of the class.
- Generally acccessed via a `get_instance()` static method.

5.50.4.3 Method Parameters

ˆ *<<**in**>>* *(p. 175)*

- An *input* parameter.

- <<***out***>> *(*p. *176)*
  - An *output* parameter.
- <<***inout***>> *(*p. *176)*
  - An *input* and *output* parameter.

5.51 .Net Language Support

One of the benefits of the Microsoft .Net platform is its support for multiple programming languages. RTI Data Distribution Service supports multiple .Net-compatible languages. The following describes how users of various languages can best use this API documentation.

5.51.1 C# Language

5.51.1.1 Namespaces

Namespace scope is indicated in the documentation with the double-colon operator (::). This syntax is for the benefit of C++/CLI users. The equivalent C# operator is the dot (.).

5.51.1.2 References

When reading this API documentation, C# users will see `'^'` and/or `'%'` notation in a number of method prototypes. For example:

```
void DomainParticipant::set_default_publisher_qos(
    PublisherQos^ qos);
void DomainParticipant::delete_publisher(
    Publisher^% publisher);
void DomainParticipant::get_default_flowcontroller_property(
    FlowControllerProperty_t% property);
```

This syntax is for the benefit of C++/CLI users. The `'^'` notation indicates a managed pointer. Since managed pointers are implicit in C#, this syntax can be ignored. The `'%'` notation is equivalent to the use of the `ref` keyword in C#.

The equivalent C# prototypes are therefore the following:

```
void DomainParticipant.set_default_publisher_qos(
    PublisherQos qos);
void DomainParticipant.delete_publisher(
    ref Publisher publisher);
void DomainParticipant.get_default_flowcontroller_property(
    ref FlowControllerProperty_t property);
```

5.51.1.3 Arrays

Arrays are expressed with the syntax `array<Foo` (p. 877)`>`, meaning a managed array of element type **Foo** (p. 877). This syntax is for the benefit of C++/CLI users. The equivalent syntax in C# is `Foo` (p. 877)`[]`.

5.51.2 C++/CLI Language

This documentation reflects the use of the API from the C++/CLI language. (This language is sometimes also referred to as "managed C++" or "C++ with managed extensions.")

5.51.3 Visual Basic .Net Language

RTI Data Distribution Service supports capabilities not offered by the Visual Basic language, such as unsigned data types. Therefore, RTI does not offer support for this language at this time.

5.52 DDS API Reference

RTI Data Distribution Service modules following the DDS module definitions.

Modules

- **Domain Module**

  *Contains the **DDS::DomainParticipant** (p. 577) class that acts as an entrypoint of RTI Data Distribution Service and acts as a factory for many of the classes. The **DDS::DomainParticipant** (p. 577) also acts as a container for the other objects that make up RTI Data Distribution Service.*

- **Topic Module**

  *Contains the **DDS::Topic** (p. 1258), **DDS::ContentFilteredTopic** (p. 419), and **DDS::MultiTopic** (p. 984) classes, the **DDS::TopicListener** (p. 1278) interface, and more generally, all that is needed by an application to define **DDS::Topic** (p. 1258) objects and attach QoS policies to them.*

- **Publication Module**

  *Contains the **DDS::FlowController** (p. 867), **DDS::Publisher** (p. 1044), and **DDS::DataWriter** (p. 499) classes as well as the **DDS::PublisherListener** (p. 1069) and **DDS::DataWriterListener** (p. 524) interfaces, and more generally, all that is needed on the publication side.*

- **Subscription Module**

  *Contains the **DDS::Subscriber** (p. 1201), **DDS::DataReader** (p. 433), **DDS::ReadCondition** (p. 1084), and **DDS::QueryCondition** (p. 1082) classes, as well as the **DDS::SubscriberListener** (p. 1226) and **DDS::DataReaderListener** (p. 461) 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.

- **Queries and Filters Syntax**

5.52.1 Detailed Description

RTI Data Distribution Service modules following the DDS module definitions.

5.52.2 Overview

Information flows with the aid of the following constructs: **DDS::Publisher** (p. 1044) and **DDS::DataWriter** (p. 499) on the sending side, **DDS::Subscriber** (p. 1201) and **DDS::DataReader** (p. 433) on the receiving side.

- A **DDS::Publisher** (p. 1044) 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. 499) object is dedicated to one application data type) accessor to a publisher. A **DDS::DataWriter** (p. 499) 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. 1201) 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. 1258) 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. 1258) 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. 499) associated with that Topic and the QoS of the **DDS::Publisher** (p. 1044) associated to the **DDS::DataWriter** (p. 499) control the behavior on the publisher's side, while the corresponding **DDS::Topic** (p. 1258), **DDS::DataReader** (p. 433) and **DDS::Subscriber** (p. 1201) 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. 1044) (or reuse an already created one) and a **DDS::DataWriter** (p. 499) with all the characteristics of the desired publication. Similarly, when an application wishes to receive data, it must cre-

ate a **DDS::Subscriber** (p. 1201) (or reuse an already created one) and a **DDS::DataReader** (p. 433) to define the subscription.

5.52.3 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. 845) supports its own specialized kind of QoS policies (see **QoS Policies** (p. 260)).

- Accepting a **DDS::Listener** (p. 952); 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. 845) supports its own specialized kind of listener. Listeners are related to changes in status conditions (see **Status Kinds** (p. 238)).

  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. 1183) (and a set of **DDS::ReadCondition** (p. 1084) objects for the **DDS::DataReader** (p. 433)); conditions (in conjunction with **DDS::WaitSet** (p. 1411) 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. 577). 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. 576) 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. 499) objects attached to the same **DDS::Publisher** (p. 1044) and retrieved among **DDS::DataReader** (p. 433) objects attached to the same **DDS::Subscriber** (p. 1201).

By definition, a **DDS::Topic** (p. 1258) corresponds to a single data type. However, several topics may refer to the same data type. Therefore, a **DDS::Topic** (p. 1258) 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. 877).

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. 1258) 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. 577).

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. 499) (this, in turn, will trigger its **DDS::Publisher** (p. 1044)).

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. 952) is used to provide a callback for synchronous access and a **DDS::WaitSet** (p. 1411) associated with one or several **DDS::Condition** (p. 408) 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. 238)). 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. 42) that the application can access as plain application data, using built-in data-readers.

5.52.4 Modules

DCPS consists of five modules:

- **Infrastructure module** (p. 107) 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. 34) contains the **DDS::DomainParticipant** (p. 577) class that acts as an entrypoint of the Service and acts as a factory for many of the classes. The **DDS::DomainParticipant** (p. 577) also acts as a container for the other objects that make up the Service.

- **Topic module** (p. 50) contains the **DDS::Topic** (p. 1258) class, the **DDS::TopicListener** (p. 1278) interface, and more generally, all that is needed by the application to define **DDS::Topic** (p. 1258) objects and attach QoS policies to them.

- **Publication module** (p. 78) contains the **DDS::Publisher** (p. 1044) and **DDS::DataWriter** (p. 499) classes as well as the **DDS::PublisherListener** (p. 1069) and **DDS::DataWriterListener** (p. 524) interfaces, and more generally, all that is needed on the publication side.

- **Subscription module** (p. 91) contains the **DDS::Subscriber** (p. 1201), **DDS::DataReader** (p. 433), **DDS::ReadCondition** (p. 1084), and **DDS::QueryCondition** (p. 1082) classes, as well as the **DDS::SubscriberListener** (p. 1226) and **DDS::DataReaderListener** (p. 461) interfaces, and more generally, all that is needed on the subscription side.

5.53 Queries and Filters Syntax

5.53.1 Syntax for DDS Queries and Filters

A subset of SQL syntax is used in several parts of the specification:

- The `filter_expression` in the **DDS::ContentFilteredTopic** (p. 419)
- The `query_expression` in the **DDS::QueryCondition** (p. 1082)
- The `topic_expression` in the **DDS::MultiTopic** (p. 984)

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*.

5.53.2 SQL grammar in BNF

```
Expression ::= FilterExpression
             | TopicExpression
             | QueryExpression
             .
FilterExpression ::= Condition
TopicExpression  ::= SelectFrom { Where } ';'
QueryExpression  ::= { Condition }{ 'ORDER BY' ( FIELDNAME // ',') }
                   .

SelectFrom       ::= 'SELECT' Aggregation 'FROM' Selection
                   .
Aggregation      ::= '*'
                   | ( SubjectFieldSpec // ',')
                   .
SubjectFieldSpec ::= FIELDNAME
```

```
              |   FIELDNAME 'AS' IDENTIFIER
              |   FIELDNAME IDENTIFIER
              .
Selection   ::= TOPICNAME
          |   TOPICNAME NaturalJoin JoinItem
          .
JoinItem    ::= TOPICNAME
          |   TOPICNAME NaturalJoin JoinItem
          |   '(' TOPICNAME NaturalJoin JoinItem ')'
          .
NaturalJoin ::= 'INNER JOIN'
          |   'INNER NATURAL JOIN'
          |   'NATURAL JOIN'
          |   'NATURAL INNER JOIN'
          .
Where       ::= 'WHERE' 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
        |   PARAMETER
        .
```

Note – INNER JOIN, INNER NATURAL JOIN, NATURAL JOIN, and NATURAL INNER JOIN are all aliases, in the sense that they have the same semantics. They are all supported because they all are part of the SQL standard.

5.53.3 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. 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"]
```

- **PARAMETER** - A parameter is of the form $\%n$, where n represents a natural number (zero included) smaller than 100. It refers to the n + 1 $^{\text{th}}$ argument in the given context. Argument can only in primitive type value format. It cannot be a FIELDNAME.

Formal notation:

```
PARAMETER : "%" (["0"-"9"])+
```

5.53.4 Type compatability in Predicate

Only certain combination of type comparisons are valid in *Predicate*. The following table marked all the compatible pairs with 'YES':

	BOOLEAN-VALUE	INTEGER-VALUE	FLOAT-VALUE	CHAR-VALUE	STRING	ENUMERATED-VALUE
BOOLEAN	YES					
INTEGERVALUE		YES	YES			
FLOAT-VALUE		YES	YES			
CHAR-VALUE				YES	YES	YES
STRING				YES	YES(∗1)	YES
ENUMERATED-VALUE		YES		YES(∗2)	YES(∗2)	YES(∗3)

ˆ (∗1) See **SQL Extension: Regular Expression Matching** (p. 189)

ˆ (∗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.

ˆ (∗3) Only for same type Enums.

5.53.5 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.

MATCH is case-sensitive.

The pattern allows limited "wild card" matching under the following rules:

Character	Meaning
,	"," separates a list of alternate patterns. The field string is matched if it matches one or more of the patterns.
/	"/" in the pattern string matches a / in the field string. This character is used to separate a sequence of mandatory substrings.
?	"?" in the pattern string matches any single *non-special* characters in the field string.
*	"*" in the pattern string matches 0 or more *non-special* characters in field string.
[***charlist***]	Matches any one of the characters from the list of characters in *charlist*.
[***s-e***]	Matches any character any character from *s* to *e*, inclusive.
%	"%" is used to designate filter expressions parameters.
[**!*charlist***] **or** [^***charlist***]	Matches any characters not in *charlist* (not supported).
[**!*s-e***] **or** [^***s-e***]	Matches any characters not in the interval [s-e] (not supported).
\	Escape character for special characters (not supported)

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.

5.53.6 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. 419)) or a **query_expression** (for **DDS::QueryCondition** (p. 1082)):

```
^  "z < 1000 AND x < 23"
```

Examples of a **filter_expression** using **MATCH** (for **DDS::ContentFilteredTopic** (p. 419)) operator:

- `"symbol MATCH 'NASDAQ/GOOG'"`
- `"symbol MATCH 'NASDAQ/[A-M]*'"`

Example of a **topic_expression** (for **DDS::MultiTopic** (p. 984) [**Not supported (optional)**]):

-
```
"SELECT flight_id, x, y, z AS height FROM 'Location' NATURAL JOIN
      'FlightPlan' WHERE height < 1000 AND x <23"
```

5.54 RTI Data Distribution Service API Reference

RTI Data Distribution Service product specific API's.

Modules

- **Clock Selection**

  APIs related to clock selection.

- **Multi-channel DataWriters**

  APIs related to Multi-channel DataWriters.

- **Pluggable Transports**

  APIs related to RTI Data Distribution Service pluggable transports.

- **Configuration Utilities**

  Utility API's independent of the DDS standard.

- **Unsupported Utilities**

  Unsupported APIs used by examples in the RTI Data Distribution Service distribution as well as in rtiddsgen-generated examples.

- **Durability and Persistence**

  APIs related to RTI Data Distribution Service Durability and Persistence.

- **Configuring QoS Profiles with XML**

  APIs related to XML QoS Profiles.

5.54.1 Detailed Description

RTI Data Distribution Service product specific API's.

5.55 Programming How-To's

These "How To"s illustrate how to apply RTI Data Distribution Service APIs to common use cases.

Modules

- **Publication Example**

  A data publication example.

- **Subscription Example**

  A data subscription example.

- **Participant Use Cases**

  Working with domain partcipants.

- **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.

- **Large Data Use Cases**

  Working with large data types.

5.55.1 Detailed Description

These "How To"s illustrate how to apply RTI Data Distribution Service APIs to common use cases.

These are a good starting point to familiarize yourself with DDS. You can use these code fragments as "templates" for writing your own code.

5.56 Programming Tools

Modules

- **rtiddsgen**

  Generates source code from data types declared in IDL, XML, XSD, or WSDL files.

- **rtiddsping**

  Sends or receives simple messages using RTI Data Distribution Service.

- **rtiddsspy**

  Debugging tool which receives all RTI Data Distribution Service communication.

5.57 rtiddsgen

Generates source code from data types declared in IDL, XML, XSD, or WSDL files. Generates code necessary to allocate, send, receive, and print user-defined data types.

5.57.1 Usage

```
rtiddsgen [-d <outdir>]
    [-language <C|C++|Java|C++/CLI|C#>]
            [-namespace]
            [-package <packagePrefix>]
            [-example <arch>]
            [-replace]
    [-debug]
            [-corba [client header file] [-orb <CORBA ORB>]]
            [-optimization <level of optimization>]
            [-stringSize <Unbounded strings size>]
            [-sequenceSize <Unbounded sequences size>]
            [-notypecode]
            [-ppDisable]
            [-ppPath <preprocessor executable>]
            [-ppOption <option>]
            [-D <name>[=<value>]]
            [-U <name>]
            [-I <directory>]
            [-noCopyable]
            [-use42eAlignment]
            [-enableEscapeChar]
            [-typeSequenceSuffix <Suffix>]
            [-convertToXml |
             -convertToXsd |
  -convertToWsdl |
  -convertToIdl]
            [-convertToCcl]
            [-convertToCcs]
            [-version]
            [-help]
            [-verbosity [1-3]]
            [[-inputIdl] <IDLInputFile.idl> |
      [-inputXml] <XMLInputFile.xml> |
[-inputXsd] <XSDInputFile.xsd> |
[-inputWsdl] <WSDLInputFile.wsdl>]
```

-d Specifies where to put the generated files. If omitted, the input file's directory is used.

-language Generates output for only the language specified. The default is C++.

-namespace Specifies the use of C++ namespaces (for C++ only).

-package Specifies a packagePrefix to use as the root package (for Java only).

-example Generates example programs and makefiles (for UNIX-based systems) or workspace and project files (for Windows systems) based on the input types description file.

The <arch> parameter specifies the architecture for the example makefiles.

For -language C/C++, valid options for <arch> are:

sparcSol2.8gcc3.2, sparcSol2.8cc5.2, sparcSol2.9gcc3.2, sparcSol2.9gcc3.3, sparcSol2.9cc5.3, sparcSol2.9cc5.4, sparc64Sol2.10cc5.8, sparcSol2.10gcc3.4.2, sparc64Sol2.10gcc3.4.2, i86Sol2.9gcc3.3.2, i86Sol2.10gcc3.4.4, x64Sol2.10gcc3.4.3

i86Linux2.4gcc3.2, i86Linux2.4gcc3.2.2, x64Linux2.4gcc3.2.3, i86Linux2.6gcc3.4.3, x64Linux2.6gcc3.4.5, i86Linux2.6gcc4.1.1, x64Linux2.6gcc4.1.1, i86Linux2.6gcc4.1.2, x64Linux2.6gcc4.1.2, x64Linux2.6gcc4.3.2, i86Linux2.6gcc3.4.6, i86RedHawk5.1gcc4.1.2, i86Suse9.5gcc3.3.3, x64Suse9.5gcc3.3.3, i86Suse10.1gcc4.1.0, x64Suse10.1gcc4.1.0, armv7leLinux2.6gcc4.4.1, ppc64Linux2.6gcc4.1.2, ppc64Linux2.6gcc4.3.0, ppc7400Linux2.6gcc3.3.3

i86Win32VC60, i86Win32VC70, i86Win32VS2003, i86Win32VS2005, x64Win64VS2005, i86Win32VS2008, x64Win64VS2008 armv4WinCE5.0VS2005, i86WinCE5.0eVC40, armv4WinCE6.0VS2005, i86WinCE6.0VS2005

ppc7400Lynx4.0.0gcc3.2.2, ppc7400Lynx4.2.0gcc3.2.2, ppc750Lynx4.0.0gcc3.2.2, ppc750Lynx4.2.0gcc3.2.2, ppc7400Lynx4.2.0gcc3.4.3, ppc7400Lynx5.0.0gcc3.4.3, i86Lynx4.0.0gcc2.95.3, i86Lynx4.0.0gcc3.2.2, i86Lynx4.2.0gcc3.2.2, i86LynxOS_SE3.0.0gcc3.4.3

ppc604Vx5.4gcc, pentiumVx5.5gcc, ppc405Vx5.5gcc, ppc604Vx5.5gcc, ppc603Vx5.5gcc, pentiumVx6.0gcc3.3.2, ppc604Vx6.0gcc3.3.2, pentiumVx6.0gcc3.3.2_rtp, ppc604Vx6.0gcc3.3.2_rtp, pentiumVx6.3gcc3.4.4, ppc604Vx6.3gcc3.4.4, pentiumVx6.3gcc3.4.4_rtp, ppc604Vx6.3gcc3.4.4_rtp, pentiumVx6.5gcc3.4.4, ppc604Vx6.5gcc3.4.4, pentiumVx6.5gcc3.4.4_rtp, ppc604Vx6.5gcc3.4.4_rtp, pentiumVx6.6gcc4.1.2, pentiumVx6.6gcc4.1.2_rtp, ppc405Vx6.6gcc4.1.2, ppc405Vx6.6gcc4.1.2_rtp, ppc604Vx6.6gcc4.1.2, ppc604Vx6.6gcc4.1.2_rtp, ppc604Vx6.7gcc4.1.2, ppc604Vx6.7gcc4.1.2_rtp,

ppc7400Inty5.0.7.mvme5100-7400, ppc7400Inty5.0.7.mvme5100-7400-ipk, ppc7400Inty5.0.9.mvme5100-7400-ghnet2, cellInty5.1.0.ibmpxcab, p5AIX5.3xlc9.0, 64p5AIX5.3xlc9.0

For -language C++/CLI and C#, valid option for <arch> is i86Win32dotnet2.0

For -language java, valid options for <arch> are:

i86Sol2.9jdk, i86Sol2.10jdk, x64Sol2.10jdk, sparcSol2.8jdk, sparcSol2.9jdk, sparcSol2.10jdk, sparc64Sol2.10jdk

i86Linux2.4gcc3.2jdk, i86Linux2.4gcc3.2.2jdk, x64Linux2.4gcc3.2.3jdk, i86Linux2.6gcc3.4.3jdk, x64Linux2.6gcc3.4.5jdk, i86Linux2.6gcc4.1.1jdk, x64Linux2.6gcc4.1.1jdk, i86Linux2.6gcc4.1.2jdk, x64Linux2.6gcc4.1.2jdk, x64Linux2.6gcc4.3.2jdk, i86Linux2.6gcc3.4.6jdk, i86RedHawk5.1gcc4.1.2jdk i86Suse9.5gcc3.3.3jdk, x64Suse9.5gcc3.3.3jdk, i86Suse10.1gcc4.1.0jdk, x64Suse10.1gcc4.1.0jdk

ppc64Linux2.6gcc4.1.2jdk, ppc64Linux2.6gcc4.3.0jdk

i86Win32jdk, x64Win64jdk

ppc7400Lynx4.0.0gcc3.2.2jdk, ppc750Lynx4.0.0gcc3.2.2jdk, ppc7400Lynx4.2.0gcc3.4.3jdk, ppc7400Lynx5.0.0gcc3.4.3jdk, i86Lynx4.0.0gcc2.95.3jdk, i86Lynx4.0.0gcc3.2.2jdk, p5AIX5.3xlc9.0jdk,64p5AIX5.3xlc9.0jdk

-replace Overwrites any existing output files. Warning: This removes any changes you may have made to the original files.

-debug Generates intermediate files for debugging purposes.

-corba [client header file] [-orb <CORBA ORB>] Specifies that you want to produce CORBA-compliant code.

Use [client header file] and [-orb <CORBA ORB>] for C++ only. The majority of code generated is independent of the ORB. However, for some IDL features, the code generated depends on the ORB. This version of rtiddsgen generates code compatible with ACE-TAO or JacORB. To pick the ACE_TAO version, use the -orb parameter; the default is ACE_TAO1.6.

client header file: the name of the header file for the IDL types generated by the CORBA IDL compiler. This file will be included in the rtiddsgen type header file instead of generating type definitions.

CORBA support requires the RTI CORBA Compatibility Kit, an add-on product that provides a different version of rtiddsgen. Please contact `support@rti.com` for more information.

-optimization Sets the optimization level. (Only applies to C/C++)

- 0 (default): No optimization.
- 1: Compiler generates extra code for typedefs but optimizes its use. If the type that is used is a typedef that can be resolved either to a primitive type or to another type defined in the same file, the generated code will invoke the code of the most basic type to which the typedef can be resolved, unless the most basic type is an array or a sequence. This level can be used if the generated code is not expected to be modified.

- 2: Maximum optimization. Functionally the same as level 1, but extra code for typedef is not generated. This level can be used if the typedefs are only referred by types within the same file.

-typeSequenceSuffix Assigns a suffix to the name of the implicit sequence defined for IDL types. (Only applies to CORBA)

By default, the suffix is 'Seq'. For example, given the type 'Foo' the name of the implicit sequence will be 'FooSeq'.

-stringSize Sets the size for unbounded strings. Default: 255 bytes.

-sequenceSize Sets the size for unbounded sequences. Default: 100 elements.

-notypecode: Disables the generation of type code information.

-ppDisable: Disables the preprocessor.

-ppPath <preprocessor executable>: Specifies the preprocessor path. If you only specify the name of an executable (not a complete path to that executable), the executable must be found in your Path.

The default value is "cpp" for non-Windows architectures, "cl.exe" for Windows architectures.

If the default preprocessor is not found in your Path and you use -ppPath to provide its full path and filename, you must also use -ppOption (described below) to set the following preprocessor options:

- If you use a non-default path for cl.exe, you also need to set:

  -ppOption /nologo -ppOption /C -ppOption /E -ppOption /X

- If you use a non-default path for cpp, you also need to set:

  -ppOption -C

-ppOption <option>: Specifies a preprocessor option. This parameter can be used multiple times to provide the command-line options for the specified preprocessor. See -ppPath (above).

-D <name>[=<value>]: Defines preprocessor macros.

-U <name>: Cancels any previous definition of name.

-I <directory>: Adds to the list of directories to be searched for type-definition files (IDL, XML, XSD or WSDL files). Note: A type-definition file in one format cannot include a file in another format.

-noCopyable: Forces rtiddsgen to put copy logic into the corresponding TypeSupport class rather than the type itself (for Java code generation only).

This option is not compatible with the use of ndds_standalone_type.jar.

-use42eAlignment: Generates code compliant with RTI Data Distribution Service 4.2e.

If your RTI Data Distribution Service applications data type uses a 'double','long long','unsigned long long', or 'long double' it will not be backwards compatible with RTI Data Distribution Service 4.2e applications, unless you use the -use42eAlignment flag when generating code with rtiddsgen.

-enableEscapeChar: Enables use of the escape character '_' in IDL identifiers. With CORBA this option is always enabled.

-convertToXml: Converts the input type-description file to XML format.

-convertToIdl: Converts the input type-description file to IDL format.

-convertToXsd: Converts the input type-description file to XSD format.

-convertToWsdl: Converts the input type-description file to WSDL format.

-convertToCcl: Converts the input type-description file to CCL format.

-convertToCcs: Converts the input type-description file to CCS format.

-version: Prints the version.

-help: Prints out this rtiddsgen usage help.

-verbosity: rtiddsgen verbosity.

- 1: exceptions
- 2: exceptions and warnings
- 3 (default): exceptions, warnings and information

-inputIdl: Indicates that the input file is an IDL file, regardless of the file extension.

-inputXml: Indicates that the input file is a XML file, regardless of the file extension.

-inputXsd: Indicates that the input file is a XSD file, regardless of the file extension.

-inputWsdl: Indicates that the input file is a WSDL file, regardless of the file extension.

IDLInputFile.idl: File containing IDL descriptions of your data types. If -inputIdl is not used, the file must have an .idl extension.

XMLInputFile.xml: File containing XML descriptions of your data types. If -inputXml is not used, the file must have an .xml extension.

XSDInputFile.xsd: File containing XSD descriptions of your data types. If -inputXsd is not used, the file must have an .xsd extension.

XSDInputFile.wsdl: WSDL file containing XSD descriptions of your data types. If -inputWsdl is not used, the file must have an .wsdl extension.

5.57.2 Description

rtiddsgen takes a language-independent specification of the data (in IDL, XML, XSD or WSDL notation) and generates supporting classes and code to distribute instances of the data over RTI Data Distribution Service.

To use rtiddsgen, you first write a description of your data types in IDL, XML, XSD or WSDL format.

5.57.3 C++ Example

The following is an example generating the RTI Data Distribution Service type myDataType:

IDL notation

```
struct myDataType {
    long value;
};
```

XML notation

```
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="rti_dds_topic_types.xsd">
    <struct name="myDataType">
        <member name="value" type="long"/>
    </struct>
</types>
```

XSD notation

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:dds="http://www.omg.org/dds"
     xmlns:tns="http://www.omg.org/IDL-Mapped/"
     targetNamespace="http://www.omg.org/IDL-Mapped/">
  <xsd:import namespace="http://www.omg.org/dds" schemaLocation="rti_dds_topic_types_common.xsd"/>
  <xsd:complexType name="myDataType">
    <xsd:sequence>
      <xsd:element name="value" minOccurs="1" maxOccurs="1" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

WSDL notation

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:dds="http://www.omg.org/dds" xmlns:x
  <types>
    <xsd:schema targetNamespace="http://www.omg.org/IDL-Mapped/">
      <xsd:import namespace="http://www.omg.org/dds" schemaLocation="rti_dds_topic_types_common.x
      <xsd:complexType name="myDataType">
        <xsd:sequence>
          <xsd:element name="value" minOccurs="1" maxOccurs="1" type="xsd:int"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </types>
</definitions>
```

Assuming the name of the idl file is myFileName.(idl|xml|xsd|wsdl) then all you need to do is type:

rtiddsgen myFileName.(idl|xml|xsd|wsdl)

This generates myFileName.cxx, myFileName.h, myFileNamePlugin.cxx, myFileNamePlugin.h, myFileNameSupport.cxx and myFileNameSupport.h. By default, rtiddsgen will not overwrite these files. You must use the -replace argument to do that.

5.57.4 IDL Language

In the IDL language, data types are described in a fashion almost identical to structures in "C." The complete description of the language can be found at the OMG website.

rtiddsgen does not support the full IDL language.

For detailed information about the IDL support in RTI Data Distribution Service see Chapter 3 of the user manual.

Below are the IDL types that are currently supported:

- char
- wchar
- octet
- short
- unsigned short
- long

- unsigned long
- long long
- unsigned long long
- float
- double
- long double
- boolean
- bounded string
- unbounded string
- bounded wstring
- unbounded wstring
- enum
- typedef
- struct
- valuetypes (limited support)
- union
- sequences
- unbounded sequences
- arrays
- array of sequences
- constant

The following non-IDL types are also supported by rtiddsgen:

- bitfield
- valued enum

Use of Unsupported Types in an IDL File

You may include unsupported data types in the IDL file. rtiddsgen does not consider this an error. This allows you to use types that are defined in non-IDL languages with either hand-written or non-rtiddsgen written plug-ins. For example, the following is allowable:

```
//@copy #include "Bar.h"
//@copy #include "BarHandGeneratedPlugin.h"
struct Foo {
short height;
Bar barMember;
};
```

In the above case, Bar is defined externally by the user.

Multiple Types in a Single File

You can specify multiple types in a single idl file. This can simplify management of files in your distributed program.

Use of Directives in an IDL File

The following directives can be used in your IDL file: Note: Do not put a space between the slashes and the @ sign. Note: Directives are case-sensitive (for example: use key, not Key).

- //@key The field declared just before this directive in the enclosing structure is part of the key. Any number of a structure's fields may be declared part of the key.
- //@copy This copies a line of text into the generated code verbatim (for all languages). The text is copied into all of the type-specific files generated by rtiddsgen (except the examples).
- //@copy-c Same as //@copy, but for C++/C-only code.
- //@copy-java Same as //@copy, but for Java-only code.
- //@copy-declaration This is like //@copy, but only copies the text into the file where the type is declared (<type>.h for C++/C, or <type>.java for Java).
- //@copy-c-declaration Same as //@copy-declaration, but for C++/C-only code.
- //@copy-java-declaration Same as //@copy-declaration, but for Java-only code.
- //@resolve-name [true|false] This specifies whether or not rtiddsgen should resolve the scope of a type. If this directive is not present or set to true, rtiddsgen resolves the scope. Otherwise rtiddsgen delegates the resolution of a type to the user.
- //@top-level [true|false] This specifies whether or not rtiddsgen should generate type-support code for a particular struct or union. The default is true.

5.57.5 XML Language

The data types can be described using XML.

RTI Data Distribution Service provides DTD and XSD files that describe the XML format.

The DTD definition of the XML elements can be found in ../../../resource/dtd/rti_dds_topic_types.dtd" under <NDDSHOME>/resource/dtd.

The XSD definition of the XML elements can be found in ../../../resource/dtd/rti_dds_topic_types.dtd" under <NDDSHOME>/resource/xsd.

The XML validation performed by rtiddsgen always uses the DTD definition. If the <!DOCTYPE> tag is not present in the XML file, rtiddsgen will look for the DTD document under <NDDSHOME>/resource/dtd. Otherwise, it will use the location specified in <!DOCTYPE>.

For detailed information about the mapping between IDL and XML see Chapter 3 of the RTI Data Distribution Service User Manual.

5.57.6 XSD Language

The data types can be described using XML schemas (XSD files). The XSD specification is based on the standard IDL to WSDL mapping described in the OMG document `CORBA to WSDL/SOAP Interworking Specification`

For detailed information about the mapping between IDL and XML see Chapter 3 of the RTI Data Distribution Service User Manual.

5.57.7 WSDL Language

The data types can be described using XML schemas contained in WSDL files. The XSD specification is based on the standard IDL to WSDL mapping described in the OMG document `CORBA to WSDL/SOAP Interworking Specification`

For detailed information about the mapping between IDL and XML see Chapter 3 of the RTI Data Distribution Service User Manual.

5.57.8 Using Generated Types Without RTI Data Distribution Service (Standalone)

You can use the generated type-specific source and header files without linking the RTI Data Distribution Service libraries or even including the RTI Data Distribution Service header files. That is, the generated files for your data types can be used standalone.

The directory <NDDSHOME>/resource/standalone contains the helper files required to work in standalone mode:

- include: header and templates files for C/C++.
- src: source files for C/C++.
- class: Java jar file.

Using Standalone Types in C

The generated files that can be used standalone are:

- <idl file name>.c : Types source file
- <idl file name>.h : Types header file

You *cannot* use the type plug-in (<idl file>Plugin.c <idl file>Plugin.h) or the type support (<idl file>Support.c <idl file>Support.h) code standalone.

To use the rtiddsgen-generated types in a standalone manner:

- Include the directory <NDDSHOME>/resource/standalone/include in the list of directories to be searched for header files.
- Add the source files ndds_standalone_type.c and <idl file name>.c to your project.
- Include the file <idl file name>.h in the source files that will use the generated types in a standalone way.
- Compile the project using the two following preprocessor definitions:
  - NDDS_STANDALONE_TYPE
  - The definition for your platform: RTI_VXWORKS, RTI_QNX, RTI_WIN32, RTI_INTY, RTI_LYNX or RTI_UNIX

Using Standalone Types in C++

The generated files that can be used standalone are:

- <idl file name>.cxx : Types source file
- <idl file name>.h : Types header file

You *cannot* use the type plugin (<idl file>Plugin.cxx <idl file>Plugin.h) or the type support (<idl file>Support.cxx <idl file>Support.h) code standalone.

To use the generated types in a standalone manner:

- Include the directory <NDDSHOME>/resource/standalone/include in the list of directories to be searched for header files.
- Add the source files ndds_standalone_type.cxx and <idl file name>.cxx to your project.
- Include the file <idl file name>.h in the source files that will use the generated types in a standalone way.
- Compile the project using the two following preprocessor definitions:
  - NDDS_STANDALONE_TYPE
  - The definition for your platform: RTI_VXWORKS, RTI_QNX, RTI_WIN32, RTI_INTY, RTI_LYNX or RTI_UNIX

Standalone Types in Java

The generated files that can be used standalone are:

- <idl type>.java
- <idl type>Seq.java

You *cannot* use the type code (<idl file>TypeCode.java), the type support (<idl type>TypeSupport.java), the data reader (<idl file>DataReader.java) or the data writer code (<idl file>DataWriter.java) standalone.

To use the generated types in a standalone manner:

- Include the file ndds_standalone_type.jar in the classpath of your project.
- Compile the project using the standalone types files (<idl type>.java <idl type>Seq.java).

5.58 rtiddsping

Sends or receives simple messages using RTI Data Distribution Service. The rtiddsping utility uses RTI Data Distribution Service to send and receive preconfigured "Ping" messages to other rtiddsping applications which can be running in the same or different computers.

The rtiddsping utility can used to test the network and/or computer configuration and the environment settings that affect the operation of RTI Data Distribution Service.

Usage

```
   rtiddsping [-help] [-version]
    [-domainId <domainId>]     ... defaults to 0
    [-index <NN>]              ... defaults to -1 (auto)
    [-appId <ID>]              ... defaults to a middleware-selected value
    [-Verbosity <NN>]          ... can be 0..5
    [-peer <PEER>]             ... PEER format is NN@TRANSPORT://ADDRESS
    [-discoveryTTL <NN>]       ... can be 0..255
[-transport <MASK>]        ... defaults to DDS_TRANSPORTBUILTIN_MASK_DEFAULT
[-msgMaxSize <SIZE>]       ... defaults to -1 (transport default)
[-shmRcvSize <SIZE>]       ... defaults to -1 (transport default)
[-deadline   <SS>]         ... defaults to -1 (no deadline)
[-durability <TYPE>]       ... TYPE can be VOLATILE or TRANSIENT_LOCAL
    [-multicast  <ADDRESS>]    ... defaults to no multicast
[-numSamples <NN>]         ... defaults to infinite
    [-publisher]                 ... this is the default
[-queueSize  <NN>]         ... defaults to 1
[-reliable]                  ... defaults to best-efforts
[-sendPeriod <SS>]         ... SS is in seconds, defaults to 1
[-subscriber]
[-timeFilter <SS>]         ... defaults to  0 (no filter)
[-timeout    <SS>]         ... SS is in seconds, defaults to infinite
[-topicName  <NAME>]       ... defaults to PingTopic
[-typeName   <NAME>]       ... defaults to PingType
[-useKeys    <NN>]         ... defaults to PingType
    [-qosFile <file>]
    [-qosProfile  <lib::prof>]
```

Example: `rtiddsping -domainId 3 -publisher -numSamples 100`

VxWorks Usage

```
   rtiddsping "[<options>]"
        The options use the same syntax as above.
```

Example `rtiddsping "-domainId 3 -publisher -numSamples 100"`

```
If the stack of the shell is not large enough to run rtiddsping, use "taskSpawn":

taskSpawn <name>,<priority>,<taskspawn options>,<stack size in bytes>,rtiddsping,"[\<options\>]"
     The options use the same syntax as above.

Example taskSpawn "rtiddsping",100,0x8,50000,rtiddsping,"-domainId 3 -publisher -numSamples 100"
```

Options:

-help Prints a help message and exits.

-version Prints the version and exits.

-Verbosity <NN> Sets the verbosity level. The range is 0 to 5.

0 has minimal output and does not echo the fact that data is being sent or received.

1 prints the most relevant statuses, including the sending and receiving of data. This is the default.

2 prints a summary of the parameters that are in use and echoes more detailed status messages.

3-5 Mostly affects the verbosity used by the internal RTI Data Distribution Service modules that implement rtiddsping. The output is not always readable; its main purpose is to provide information that may be useful to RTI's support team.

Example: rtiddsping -Verbosity 2

-domainId <NN>

Sets the domain ID. The valid range is 0 to 100.

Example: rtiddsping -domainId 31

-appId <ID>

Sets the application ID. If unspecified, the system will pick one automatically.

This option is rarely used.

Example: rtiddsping -appId 34556

-index <NN>

Sets the participantIndex. If participantIndex is not -1 (auto), it must be different than the one used by all other applications in the same computer and domainId. If this is not respected, rtiddsping (or the application that starts last) will get an initialization error.

Example: rtiddsping -index 2

-peer <PEER>

Specifies a PEER to be used for discovery. Like any RTI Data Distribution Service application, it defaults to the setting of the environment variable NDDS_DISCOVERY_PEERS or a preconfigured multicast address if the environment is not set.

The format used for PEER is the same one used for NDDS_DISCOVERY_PEERS and is described in detail in **NDDS_DISCOVERY_PEERS** (p. 312). A brief summary follows:

The general format is: NN@TRANSPORT://ADDRESS where:

- ADDRESS is an address (in name form or using the IP notation xxx.xxx.xxx.xxx). ADDRESS may be a multicast address.
- TRANSPORT represents the kind of transport to use and NN is the maximum participantIndex expected at that location. NN can be omitted and it is defaulted to '4'
- Valid settings for TRANSPORT are 'udpv4' and 'shmem'. The default setting if the transport is omitted is 'udpv4'.
- ADDRESS cannot be omitted if the '-peer' option is specified.

The -peer option may be repeated to specify multiple peers.

Example: rtiddsping -peer 10.10.1.192 -peer mars -peer 4@pluto

-discoveryTTL <TTL>

Sets the TTL (time-to-live) used for multicast discovery. If not specified, it defaults to the built-in RTI Data Distribution Service default.

The valid range is 0 to 255. The value '0' limits multicast to the node itself (i.e., can only discover applications running on the same computer). The value '1' limits multicast discovery to computers on the same subnet. Values higher than 1 generally indicate the maximum number of routers that may be traversed (although some routers may be configured differently).

Example: rtiddsping -discoveryTTL 16

-transport <MASK>

A bit-mask that sets the enabled builtin transports. If not specified, the default set of transports is used (UDPv4 + shmem). The bit values are: 1=UDPv4, 2=shmem, 8=UDPv6.

-msgMaxSize <SIZE>

Configure the maximum message size allowed by the installed transports. This is needed if you are using rtiddsping to communicate with an application that has set these transport parameters to larger than default values.

-shmRcvSize <SIZE>

Increase the shared memory receive-buffer size. This is needed if you are using rtiddsping to communicate with an application that has set these transport parameters to larger than default values.

-deadline <SS>

This option only applies if the '-subscriber' option is also specified.

Sets the DEADLINE QoS for the subscriptions made by rtiddsping.

Note that this may cause the subscription QoS to be incompatible with the publisher if the publisher did not specify a sendPeriod greater than the deadline. If the QoS is incompatible, rtiddsping will not receive updates.

Each time a deadline is detected, rtiddsping will print a message indicating the number of deadlines received so far.

Example: rtiddsping -deadline 3.5

-durability <TYPE>

Sets the DURABILITY QoS used for publishing or subscribing. Valid settings are: VOLATILE and TRANSIENT_LOCAL (default). The effect of this setting can only be observed when it is used in in conjunction with reliability and a queueSize larger than 1. If all these conditions are met, a late-joining subscriber will be able to see up to queueSize samples that were previously written by the publisher.

Example: rtiddsping -durability VOLATILE

-multicast <ADDRESS>

This option only applies if the '-subscriber' option is also specified.

Configures ping to receive messages over multicast. The <ADDRESS> parameter indicates the address to use. ADDRESS must be in the valid range for multicast addresses. For IP version 4 the valid range is 224.0.0.1 to 239.255.255.255

Example: rtiddsping -multicast 225.1.1.1

-numSamples <NN>

Sets the number of samples that will be sent by rtiddsping. After those samples are sent, rtiddsping will exit. messages.

Example: rtiddsping -numSamples 10

-publisher

Causes rtiddsping to send ping messages. This is the default.

Example: rtiddsping -publisher

-queueSize <NN>

Specifies the maximal number of samples to hold in the queue. In the case of the publisher, it affects the samples that are available for a late-joining subscriber.

Example: rtiddsping -queueSize 100

-reliable

Configures the RELIABILITY QoS for publishing or subscribing. The default setting (if -reliable is not used) is BEST_EFFORT

Example: rtiddsping -reliable

-sendPeriod <SS>

Sets the period (in seconds) at which rtiddsping sends the messages.

Example: rtiddsping -sendPeriod 0.5

-subscriber

Causes rtiddsping to listen for ping messages. This option cannot be specified if '-publisher' is also specified.

Example: rtiddsping -subscriber

-timeFilter <SS>

This option only applies if the '-subscriber' option is also specified.

Sets the TIME_BASED_FILTER QoS for the subscriptions made by rtiddsping. This QoS causes RTI Data Distribution Service to filter out messages that are published at a rate faster than what the filter duration permits. For example, if the filter duration is 10 seconds, messages will be printed no faster than once every 10 seconds.

Example: rtiddsping -timeFilter 5.5

-timeout <SS>

This option only applies if the '-subscriber' option is also specified.

Sets a timeout (in seconds) that will cause rtiddsping to exit if no samples are received for a duration that exceeds the timeout.

Example: rtiddsping -timeout 30

-topicName <NAME>

Sets the topic name used by rtiddsping. The default is 'RTIddsPingTopic'. To communicate, both the publisher and subscriber must specify the same topic name.

Example: rtiddsping -topicName Alarm

-typeName <NAME>

Sets the type name used by rtiddsping. The default is 'RTIddsPingType'. To communicate, both publisher and subscriber must specify the same type name.

Example: rtiddsping -typeName AlarmDescription

-useKeys <NN>

This option causes rtiddsping to use a topic whose data contains a key. The value of the NN parameter indicates the number of different data objects (each identified by a different value of the key) that will be published by rtiddsping. The value of NN only affects the publishing behavior. However NN still needs to be specified when the -useKeys option is used with the -subscriber option.

For communication to occur, both the publisher and subscriber must agree on whether the topic that they publish/subscribe contains a key. Consequently, if you specify the -useKeys parameter for the publisher, you must do the same with the subscriber. Otherwise communication will not be stablished.

Example: rtiddsping -useKeys 20

-qosFile <file>

Allow you to specify additional QoS XML settings using url_profile. For more information on the syntax, see Chapter 15 in the RTI Data Distribution Service User's Manual.

Example: rtiddsping -qosFile /home/user/QoSProfileFile.xml

-qosProfile <lib::prof>

This option specifies the library name and profile name that the tool should use.

QoS settings

rtiddsping is configured internally using a special set of QoS settings in a profile called InternalPingLibrary::InternalPingProfile. This is the default profile unless a profile called DefaultPingLibrary::DefaultPingProfile is found. You can use the command-line option -qosProfile to tell rtiddsping to use a different lib::profile instead of DefaultPingLibrary::DefaultPingProfile. Like all the other RTI Data Distribution Service applications, rtiddsping loads all the profiles specified using the environment variable NDDS_QOS_PROFILES or the file named USER_QOS_PROFILES found in the current working directory.

The QoS settings used internally are available in the file RTIDDSSPING_QOS_PROFILES.example.xml.

Description

The usage depends on the operating system from which rtiddsping is executed.

Examples for UNIX, Linux, and Windows Systems

On UNIX, Linux, Windows and other operating systems that have a shell, the syntax matches the one of the regular commands available in the shell. In the examples below, the string 'shell prompt>' represents the prompt that the shell prints and are not part of the command that must be typed.

```
shell prompt>  rtiddsping -domainId 3 -publisher -numSamples 100
shell prompt>  rtiddsping -domainId 5 -subscriber -timeout 20
```

```
shell prompt>  rtiddsping -help
```

VxWorks examples:

On VxWorks systems, the libraries libnddscore.so, libnddsc.so and libnddscpp.so must first be loaded. The rtiddsping command must be typed to the VxWorks shell (either an rlogin shell, a target-server shell, or the serial line prompt). The arguments are passed embedded into a single string, but otherwise have the same syntax as for Unix/Windows. In the Unix, Linux, Windows and other operating systems that have a shell, the syntax matches the one of the regular commands available in the shell. In the examples below, the string 'vxworks prompt>' represents the prompt that the shell prints and are not part of the command that must be typed.

```
vxworks prompt>  rtiddsping "-domainId 3 -publisher -numSamples 100"
vxworks prompt>  rtiddsping "-domainId 5 -subscriber -timeout 20"
vxworks prompt>  rtiddsping "-help"
```

```
or, alternatively (to avoid overflowing the stack):
```

```
vxworks prompt> taskSpawn "rtiddsping", 100, 0x8, 50000, rtiddsping, "-domainId 3 -publisher -nu
vxworks prompt> taskSpawn "rtiddsping", 100, 0x8, 50000, rtiddsping, "-domainId 5 -subscriber -t
vxworks prompt> taskSpawn "rtiddsping", 100, 0x8, 50000, rtiddsping, "-help"
```

5.59 rtiddsspy

Debugging tool which receives all RTI Data Distribution Service communication. The rtiddsspy utility allows the user to monitor groups of publications available on any RTI Data Distribution Service domain.

Note: If you have more than one DataWriter for the same Topic, and these DataWriters have different settings for the Ownership QoS, then rtiddsspy will only receive (and thus report on) the samples from the first DataWriter.

To run rtiddsspy, like any RTI Data Distribution Service application, you must have the NDDS_DISCOVERY_PEERS environment variable that defines your RTI Data Distribution Service domain; otherwise you must specify the peers as command line parameters.

Usage

```
rtiddsspy [-help] [-version]
     [-domainId <domainId>]     ... defaults to 0
     [-index <NN>]              ... defaults to -1 (auto)
     [-appId <ID>]              ... defaults to a middleware-selected value
     [-Verbosity <NN>]          ... can be 0..5
     [-peer <PEER>]             ... PEER format is NN@TRANSPORT://ADDRESS
     [-discoveryTTL <NN>]       ... can be 0..255
     [-transport <MASK>]        ... defaults to DDS_TRANSPORTBUILTIN_MASK_DEFAULT
     [-msgMaxSize <SIZE>]       ... defaults to -1 (transport default)
     [-shmRcvSize <SIZE>]       ... defaults to -1 (transport default)
     [-tcMaxSize <SIZE>]        ... defaults to 4096
     [-hOutput]
     [-deadline   <SS>]         ... defaults to -1 (no deadline)
     [-history   <DEPTH>]       ... defaults to 8192
     [-timeFilter <SS>]         ... defaults to  0 (no filter)
     [-useFirstPublicationQos]
     [-showHandle]
     [-typeRegex <REGEX>]       ... defaults to "*"
     [-topicRegex <REGEX>]      ... defaults to "*"
     [-typeWidth <WIDTH>]       ... can be 1..255
     [-topicWidth <WIDTH>]      ... can be 1..255
     [-truncate]
     [-printSample]
     [-qosFile <file>]
     [-qosProfile  <lib::prof>]
```

Example: `rtiddsspy -domainId 3 -topicRegex "Alarm*"`

VxWorks Usage

```
rtiddsspy "[<options>]"
```

```
      The options use the same syntax as above.
```

Example `rtiddsspy "-domainId 3 -topicRegex Alarm*"`

```
rtiddsspy requires about 25 kB of stack. If the stack size of the shell from which it is invok
```

taskSpawn `<name>, <priority>, <taskspawn options>, <stack size in bytes>, rtiddsspy, "[\<opt`

```
The options use the same syntax as above.
```

Example `taskSpawn "rtiddsspy", 100, 0x8, 50000, rtiddsspy, "-domainId 3 -topicRegex Alarm*"`

Options:

-help Prints a help message and exits.

-version Prints the version and exits.

-Verbosity <NN> Sets the verbosity level. The range is 0 to 5.

0 has minimal output and does not echo the fact that data is being sent or received.

1 prints the most relevant statuses, including the sending and receiving of data. This is the default.

2 prints a summary of the parameters being used and echoes more detailed status messages.

3-5 Mostly affect the verbosity used by the internal RTI Data Distribution Service modules that implement rtiddsspy. The output is not always readable; its main purpose is to provide information that may be useful to RTI's support team.

Example: rtiddsspy -Verbosity 2

-domainId <NN>

Sets the domain ID. The valid range is 0 to 100.

Example: rtiddsspy -domainId 31

-appId <ID>

Sets the application ID. If unspecified, the system will pick one automatically.

This option is rarely used.

Example: rtiddsspy -appId 34556

-index <NN>

Sets the participantIndex. If participantIndex is not -1 (auto), it must be different than the one used by all other applications in the same computer and

domainId. If this is not respected, rtiddsspy (or the application that starts last) will get an initialization error.

Example: rtiddsspy -index 2

-peer <PEER>

Specifies a PEER to be used for discovery. Like any RTI Data Distribution Service application, it defaults to the setting of the environment variable NDDS_DISCOVERY_PEERS or a preconfigured multicast address if the environment is not set.

The format used for PEER is the same used for the NDDS_DISCOVERY_PEERS and is described in detail in **NDDS_DISCOVERY_PEERS** (p. 312). A brief summary follows:

The general format is: NN@TRANSPORT://ADDRESS where:

- ADDRESS is an address (in name form or using the IP notation xxx.xxx.xxx.xxx). ADDRESS may be a multicast address.
- TRANSPORT represents the kind of transport to use and NN is the maximum participantIndex expected at that location. NN can be omitted and it is defaulted to '4'
- Valid settings for TRANSPORT are 'udpv4' and 'shmem'. The default setting if the transport is omitted is 'udpv4'
- ADDRESS cannot be omitted if the '-peer' option is specified.

The -peer option may be repeated to specify multiple peers.

Example: rtiddsspy -peer 10.10.1.192 -peer mars -peer 4@pluto

-discoveryTTL <TTL>

Sets the TTL (time-to-live) used for multicast discovery. If not specified, it defaults to the built-in RTI Data Distribution Service default.

The valid range is 0 to 255. The value '0' limits multicast to the node itself (i.e. can only discover applications running on the same computer). The value '1' limits multicast discovery to computers on the same subnet. Settings greater than 1 generally indicate the maximum number of routers that may be traversed (although some routers may be configured differently).

Example: rtiddsspy -discoveryTTL 16

-transport <MASK>

SPecifies a bit-mask that sets the enabled builtin transports. If not specified, the default set of transports is used (UDPv4 + shmem). The bit values are: 1=UDPv4, 2=shmem, 8=UDPv6.

-msgMaxSize <SIZE>

Configures the maximum message size allowed by the installed transports. This is needed if you are using rtiddsspy to communicate with an application that has set these transport parameters to larger than default values.

-shmRcvSize <SIZE>

Increases the shared memory receive-buffer size. This is needed if you are using rtiddsspy to communicate with an application that has set these transport parameters to larger than default values.

-tcMaxSize <SIZE>

Configures the maximum size, in bytes, of a received type code.

-hOutput

Prints information on the output format used by rtiddsspy.

This option prints an explanation of the output and then exits.

Example: rtiddsspy -hOutput

-deadline <SS>

Sets the requested DEADLINE QoS for the subscriptions made by rtiddsspy.

Note that this may cause the subscription QoS to be incompatible with the publisher if the publisher did not specify an offered deadline that is greater or equal to the one requested by rtiddsspy. If the QoS is incompatible rtiddsspy will not receive updates from that writer.

Each time a deadline is detected rtiddsspy will print a message that indicates the number of deadlines received so far.

Example: rtiddsspy -deadline 3.5

-timeFilter <SS>

Sets the TIME_BASED_FILTER QoS for the subscriptions made by rtiddsspy. This QoS causes RTI Data Distribution Service to filter-out messages that are published at a rate faster than what the filter duration permits. For example if the filter duration is 10 seconds, messages will be printed no faster than once each 10 seconds.

Example: rtiddsspy -timeFilter 10.0

-history <DEPTH>

Sets the HISTORY depth QoS for the subscriptions made by rtiddsspy.

This may be relevant if the publisher has batching turned on, or if the -useFirstPublicationQos option is used that is causing a reliable or durable subscription to be created.

Example: rtiddsspy -history 1

-useFirstPublicationQos

Sets the RELIABILITY and DURABILITY QoS of the subscription based on the first discovered publication of that topic.

See also -history option.

Example: rtiddsspy -useFirstPublicationQos

-showHandle

Prints additional information about each sample received. The additional information is the 'instance_handle' field in the SampleHeader, which can be used to distinguish among multiple instances of data objects published under the same topic and type names.

Samples displayed that share the topic and type names and also have the same value for the instance_handle represent value updates to the same data object. On the other hand, samples that share the topic and type names but display different values for the instance_handle.

This option causes rtiddsspy to print an explanation of updates to the values of different data objects.

Example: rtiddsspy -showHandle

-typeRegex <REGEX>

Subscribe only to types that match the REGEX regular expression. The syntax of the regular expression is defined by the POSIX regex function.

When typing a regular expression to a command-line shell, some symbols may need to be escaped to avoid interpretation by the shell. In general, it is safest to include the expression in double quotes.

This option may be repeated to specify multiple topic expressions.

Example: rtiddsspy -typeRegex "SensorArray*"

-topicRegex <REGEX>

Subscribe only to topics that match the REGEX regular expression. The syntax of the regular expression is defined by the POSIX regex function.

When typing a regular expression to a command-line shell, some symbols may need to be escaped to avoid interpretation by the shell. In general, it is safest to include the expression in double quotes.

This option may be repeated to specify topic multiple expressions.

Example: rtiddsspy -topicRegex "Alarm*"

-typeWidth <WIDTH>

Sets the maximum width of the Type name column. Names wider than this will wrap around, unless `-truncate` is specified. Can be 1..255.

-topicWidth <WIDTH>

Sets the maximum width of the Topic name column. Names wider than this will wrap around, unless `-truncate` is specified. Can be 1..255.

-truncate

Specifies that names exceeding the maximum number of characters should be truncated.

-printSample

Prints the value of the received samples.

-qosFile <file>

Allows you to specify additional QoS XML settings using url_profile. For more information on the syntax, see Chapter 15 in the RTI Data Distribution Service User's Manual.

Example: rtiddsspy -qosFile /home/user/QoSProfileFile.xml

-qosProfile <lib::prof>

Specifies the library name and profile name to be used.

QoS settings

rtiddsspy is configured to discover as many entities as possible. To do so, an internal profile is defined, called InternalSpyLibrary::InternalSpyProfile. This is the default profile, unless a profile called DefaultSpyLibrary::DefaultSpyProfile is found. You can use the command-line option -qosProfile to tell rtiddsspy to use a speficied lib::profile instead of DefaultSpyLibrary::DefaultSpyProfile. Like all the other RTI Data Distribution Service applications, rtiddsspy loads all the profiles specified using the environment variable NDDS_QOS_PROFILES or the file named USER_QOS_PROFILES found in the current working directory.

The QoS settings used internally are available in the file RTIDDSSPY_QOS_PROFILES.example.xml.

Usage Examples

The usage depends on the operating system from which rtiddsspy is executed.

Examples for UNIX, Linux, Windows systems

On UNIX, Linux, Windows and other operating systems that have a shell, the syntax matches the one of the regular commands available in the shell. In the examples below, the string 'shell prompt>' represents the prompt that the shell prints and are not part of the command that must be typed.

```
shell prompt>  rtiddsspy -domainId 3
shell prompt>  rtiddsspy -domainId 5 -topicRegex "Alarm*"
shell prompt>  rtiddsspy -help
```

Examples for VxWorks Systems

On VxWorks systems, the libraries libnddscore.so, libnddsc.so and libnddscpp.so must first be loaded. The rtiddsspy command must be typed to the VxWorks shell (either an rlogin shell, a target-server shell, or the serial line prompt). The arguments are passed embedded into a single string, but otherwise have the same syntax as for Unix/Windows. In UNIX, Linux, Windows and other operating systems that have a shell, the syntax matches the one of the regular comamnds available in the shell. In the examples below, the string 'vxworks prompt>' represents the prompt that the shell prints and are not part of the command that must be typed.

```
vxworks prompt>  rtiddsspy "-domainId 3"
vxworks prompt>  rtiddsspy "-domainId 5  5 -topicRegex "Alarm*"
vxworks prompt>  rtiddsspy "-help"
```

5.60 Octets Built-in Type

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

Classes

- struct **DDS::Bytes**

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

- class **DDS::BytesSeq**

  Instantiates `DDS::Sequence` (p. 1163) < ***DDS::Bytes*** *(p. 388)* > .

- class **DDS::BytesTypeSupport**

  <<**interface**>> (p. 175) ***DDS::Bytes*** *(p. 388) type support.*

- class **DDS::BytesDataReader**

  <<**interface**>> (p. 175) *Instantiates* `DataReader` (p. 433) < ***DDS::Bytes*** *(p. 388)* >.

- class **DDS::BytesDataWriter**

  <<**interface**>> (p. 175) *Instantiates* `DataWriter` (p. 499) < ***DDS::Bytes*** *(p. 388)* >.

5.60.1 Detailed Description

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

5.61 KeyedOctets Built-in Type

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

Classes

- ^ struct **DDS::KeyedBytes**

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

- ^ class **DDS::KeyedBytesSeq**

  Instantiates `DDS::Sequence` (p. 1163) < ***DDS::KeyedBytes*** *(p. 915)* >.

- ^ class **DDS::KeyedBytesTypeSupport**

  <<**interface**>> (p. 175) ***DDS::KeyedBytes*** *(p. 915) type support.*

- ^ class **DDS::KeyedBytesDataReader**

  <<**interface**>> (p. 175) *Instantiates* `DataReader` (p. 433) < ***DDS::KeyedBytes*** *(p. 915)* >.

- ^ class **DDS::KeyedBytesDataWriter**

  <<**interface**>> (p. 175) *Instantiates* `DataWriter` (p. 499) < ***DDS::KeyedBytes*** *(p. 915)* >.

5.61.1 Detailed Description

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

5.62 KeyedString Built-in Type

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

Classes

- struct **DDS::KeyedString**

  Keyed string built-in type.

- class **DDS::KeyedStringSeq**

  Instantiates `DDS::Sequence` (p. 1163) < ***DDS::KeyedString*** *(p. 933)* > .

- class **DDS::KeyedStringTypeSupport**

  <<**interface**>> (p. 175) *Keyed string type support.*

- class **DDS::KeyedStringDataReader**

  <<**interface**>> (p. 175) *Instantiates* `DataReader` (p. 433) < ***DDS::KeyedString*** *(p. 933)* >.

- class **DDS::KeyedStringDataWriter**

  <<**interface**>> (p. 175) *Instantiates* `DataWriter` (p. 499) < ***DDS::KeyedString*** *(p. 933)* >.

5.62.1 Detailed Description

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

5.63 String Built-in Type

Built-in type consisting of a single character string.

Classes

- ˆ class **DDS::StringTypeSupport**

  <<**interface**>> (p. 175) *String type support.*

- ˆ class **DDS::StringDataReader**

  <<**interface**>> (p. 175) *Instantiates* `DataReader` (p. 433) < *System::String* >.

- ˆ class **DDS::StringDataWriter**

  <<**interface**>> (p. 175) *Instantiates* `DataWriter` (p. 499) < *System::String* >.

5.63.1 Detailed Description

Built-in type consisting of a single character string.

5.64 Participant Built-in Topics

Builtin topic for accessing information about the DomainParticipants discovered by RTI Data Distribution Service.

Classes

- ^ class **DDS::ParticipantBuiltinTopicData**

  *Entry created when a **DomainParticipant** (p. 577) object is discovered.*

- ^ class **DDS::ParticipantBuiltinTopicDataSeq**

  Instantiates `DDS::Sequence` (p. 1163) < ***DDS::ParticipantBuiltinTopicData*** *(p. 1002)* > .

- ^ class **DDS::ParticipantBuiltinTopicDataTypeSupport**

  Instantiates `TypeSupport` (p. 1385) < ***DDS::ParticipantBuiltinTopicData*** *(p. 1002)* > .

- ^ class **DDS::ParticipantBuiltinTopicDataDataReader**

  Instantiates `DataReader` (p. 433) < ***DDS::ParticipantBuiltinTopicData*** *(p. 1002)* > .

Properties

- ^ static System::String^ **DDS::ParticipantBuiltinTopicDataTypeSupport::PARTICIPANT_TOPIC_NAME** `[get]`

  Participant topic name.

5.64.1 Detailed Description

Builtin topic for accessing information about the DomainParticipants discovered by RTI Data Distribution Service.

5.64.2 Properties

5.64.2.1 System:: String^ DDS::ParticipantBuiltinTopicDataTypeSupport::PARTICIPANT_TOPIC_NAME `[static, get, inherited]`

Participant topic name.

Topic (p. 1258) name of **DDS::ParticipantBuiltinTopicDataDataReader** (p. 1005)

See also:

DDS::ParticipantBuiltinTopicData (p. 1002)
DDS::ParticipantBuiltinTopicDataDataReader (p. 1005)

5.65 Topic Built-in Topics

Builtin topic for accessing information about the Topics discovered by RTI Data Distribution Service.

Classes

- class **DDS::TopicBuiltinTopicData**

  Entry created when a ***Topic*** *(p. 1258) object discovered.*

- class **DDS::TopicBuiltinTopicDataSeq**

  Instantiates `DDS::Sequence` (p. 1163) < ***DDS::TopicBuiltinTopicData*** *(p. 1268)* > .

- class **DDS::TopicBuiltinTopicDataTypeSupport**

  Instantiates `TypeSupport` (p. 1385) < ***DDS::TopicBuiltinTopicData*** *(p. 1268)* > .

- class **DDS::TopicBuiltinTopicDataDataReader**

  Instantiates `DataReader` (p. 433) < ***DDS::TopicBuiltinTopicData*** *(p. 1268)* > .

Properties

- static System::String^ **DDS::TopicBuiltinTopicDataTypeSupport::TOPIC_-TOPIC_NAME** `[get]`

  Topic *(p. 1258) topic name.*

5.65.1 Detailed Description

Builtin topic for accessing information about the Topics discovered by RTI Data Distribution Service.

5.65.2 Properties

5.65.2.1 System:: String^ DDS::TopicBuiltinTopicDataTypeSupport::TOPIC_-TOPIC_NAME `[static, get, inherited]`

Topic (p. 1258) topic name.

Topic (p. 1258) name of **DDS::TopicBuiltinTopicDataDataReader** (p. 1273)

See also:

DDS::TopicBuiltinTopicData (p. 1268)
DDS::TopicBuiltinTopicDataDataReader (p. 1273)

5.66 Publication Built-in Topics

Builtin topic for accessing information about the Publications discovered by RTI Data Distribution Service.

Classes

- ^ class **DDS::PublicationBuiltinTopicData**

  *Entry created when a **DDS::DataWriter** (p. 499) is discovered in association with its **Publisher** (p. 1044).*

- ^ class **DDS::PublicationBuiltinTopicDataSeq**

  Instantiates `DDS::Sequence` (p. 1163) < ***DDS::PublicationBuiltinTopicData*** (p. 1030) > .

- ^ class **DDS::PublicationBuiltinTopicDataTypeSupport**

  Instantiates `TypeSupport` (p. 1385) < ***DDS::PublicationBuiltinTopicData*** (p. 1030) > .

- ^ class **DDS::PublicationBuiltinTopicDataDataReader**

  Instantiates `DataReader` (p. 433) < ***DDS::PublicationBuiltinTopicData*** (p. 1030) > .

Properties

- ^ static System::String^ **DDS::PublicationBuiltinTopicDataTypeSupport::PUBLICATION_TOPIC_NAME** `[get]`

  Publication topic name.

5.66.1 Detailed Description

Builtin topic for accessing information about the Publications discovered by RTI Data Distribution Service.

5.66.2 Properties

5.66.2.1 System:: String^ DDS::PublicationBuiltinTopicDataTypeSupport::PUBLICATION_TOPIC_NAME [static, get, inherited]

Publication topic name.

Topic (p. 1258) name of **DDS::PublicationBuiltinTopicDataDataReader** (p. 1038)

See also:

DDS::PublicationBuiltinTopicData (p. 1030)
DDS::PublicationBuiltinTopicDataDataReader (p. 1038)

5.67 Subscription Built-in Topics

Builtin topic for accessing information about the Subscriptions discovered by RTI Data Distribution Service.

Classes

- class **DDS::SubscriptionBuiltinTopicData**

  *Entry created when a **DDS::DataReader** (p. 433) is discovered in association with its **Subscriber** (p. 1201).*

- class **DDS::SubscriptionBuiltinTopicDataSeq**

  Instantiates `DDS::Sequence` (p. 1163) < ***DDS::SubscriptionBuiltinTopicData*** (p. 1233) > .

- class **DDS::SubscriptionBuiltinTopicDataTypeSupport**

  Instantiates `TypeSupport` (p. 1385) < ***DDS::SubscriptionBuiltinTopicData*** (p. 1233) > .

- class **DDS::SubscriptionBuiltinTopicDataDataReader**

  Instantiates `DataReader` (p. 433) < ***DDS::SubscriptionBuiltinTopicData*** (p. 1233) > .

Properties

- static System::String^ **DDS::SubscriptionBuiltinTopicDataTypeSupport::SUBSCRIPTIC**
  TOPIC_NAME `[get]`

  Subscription topic name.

5.67.1 Detailed Description

Builtin topic for accessing information about the Subscriptions discovered by RTI Data Distribution Service.

5.67.2 Properties

5.67.2.1 System:: String^ DDS::SubscriptionBuiltinTopicDataTypeSupport::SUBSCRIPTION_-TOPIC_NAME [static, get, inherited]

Subscription topic name.

Topic (p. 1258) name of **DDS::SubscriptionBuiltinTopicDataDataReader** (p. 1241)

See also:

DDS::SubscriptionBuiltinTopicData (p. 1233)
DDS::SubscriptionBuiltinTopicDataDataReader (p. 1241)

5.68 Return Codes

Types of return codes.

Classes

- class **DDS::Exception**

  Superclass of all exceptions thrown by the RTI Data Distribution Service API.

- class **DDS::Retcode_Error**

  Generic, unspecified error.

- class **DDS::Retcode_Unsupported**

  Unsupported operation. Can only returned by operations that are unsupported.

- class **DDS::Retcode_BadParameter**

  Illegal parameter value.

- class **DDS::Retcode_PreconditionNotMet**

  A pre-condition for the operation was not met.

- class **DDS::Retcode_OutOfResources**

  RTI Data Distribution Service ran out of the resources needed to complete the operation.

- class **DDS::Retcode_NotEnabled**

  *Operation invoked on a **DDS::Entity** (p. 845) that is not yet enabled.*

- class **DDS::Retcode_ImmutablePolicy**

  Application attempted to modify an immutable QoS policy.

- class **DDS::Retcode_InconsistentPolicy**

  Application specified a set of QoS policies that are not consistent with each other.

- class **DDS::Retcode_AlreadyDeleted**

  The object target of this operation has already been deleted.

- class **DDS::Retcode_Timeout**

  The operation timed out.

ˆ class **DDS::Retcode_NoData**

Indicates a transient situation where the operation did not return any data but there is no inherent error.

ˆ class **DDS::Retcode_IllegalOperation**

The operation was called under improper circumstances.

Enumerations

ˆ 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 }

Superclass of all exceptions thrown by the RTI Data Distribution Service API.

5.68.1 Detailed Description

Types of return codes.

5.68.2 Standard Return Codes

Any operation with return type **DDS::Exception** (p. 861) may return DDS::Exception::RETCODE_OK **DDS::Retcode_Error** (p. 1116) or **DDS::Retcode_IllegalOperation** (p. 1117). Any operation that takes one or more input parameters may additionally return **DDS::Retcode_BadParameter** (p. 1115). Any operation on an object created from any

of the factories may additionally return **DDS::Retcode_AlreadyDeleted** (p. 1114). Any operation that is stated as optional may additionally return **DDS::Retcode_Unsupported** (p. 1125).

Thus, the standard return codes are:

- **DDS::Retcode_Error** (p. 1116)
- **DDS::Retcode_IllegalOperation** (p. 1117)
- **DDS::Retcode_AlreadyDeleted** (p. 1114)
- **DDS::Retcode_BadParameter** (p. 1115)
- **DDS::Retcode_Unsupported** (p. 1125)

Operations that may return any of the additional return codes will state so explicitly.

5.68.3 Enumeration Type Documentation

5.68.3.1 enum DDS::ReturnCode_t

Superclass of all exceptions thrown by the RTI Data Distribution Service API.

Applications are not expected to throw or extend this type, but to handle exceptions of its more-specific subclasses.

Enumerator:

RETCODE_OK Successful return.

RETCODE_ERROR Generic, unspecified error.

RETCODE_UNSUPPORTED Unsupported operation. Can only returned by operations that are unsupported.

RETCODE_BAD_PARAMETER Illegal parameter value.

The value of the parameter that is passed in has llegal value. Things that falls into this category includes null parameters and parameter values that are out of range.

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.

RETCODE_OUT_OF_RESOURCES RTI Data Distribution Service ran out of the resources needed to complete the operation.

RETCODE_NOT_ENABLED Operation invoked on a **DDS::Entity** (p. 845) that is not yet enabled.

RETCODE_IMMUTABLE_POLICY Application attempted to modify an immutable QoS policy.

RETCODE_INCONSISTENT_POLICY Application specified a set of QoS policies that are not consistent with each other.

RETCODE_ALREADY_DELETED The object target of this operation has already been deleted.

RETCODE_TIMEOUT The operation timed out.

RETCODE_NO_DATA Indicates a transient situation where the operation did not return any data but there is no inherent error.

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_PreconditionNotMet** (p. 1123), except that there is no precondition that could be changed to make the operation succeed.

5.69 Status Kinds

Kinds of communication status.

Enumerations

ˆ enum **DDS::StatusMask** {

DDS::STATUS_MASK_NONE,

DDS::STATUS_MASK_ALL }

A bit-mask (list) of concrete status types, i.e. DDS::StatusKind[].

ˆ 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::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 }

Type for status *kinds.*

5.69.1 Detailed Description

Kinds of communication status.

Entity:

DDS::Entity (p. 845)

QoS:

QoS Policies (p. 260)

Listener:

DDS::Listener (p. 952)

Each concrete **DDS::Entity** (p. 845) is associated with a set of Status objects whose value represents the communication status of that entity. Each status value can be accessed with a corresponding method on the **DDS::Entity** (p. 845).

When these status values change, the corresponding **DDS::StatusCondition** (p. 1183) objects are activated and the proper **DDS::Listener** (p. 952) objects are invoked to asynchronously inform the application.

An application is notified of communication status by means of the **DDS::Listener** (p. 952) or the **DDS::WaitSet** (p. 1411) / **DDS::Condition** (p. 408) mechanism. The two mechanisms may be combined in the application (e.g., using **DDS::WaitSet** (p. 1411) (s) / **DDS::Condition** (p. 408) (s) to access the data and **DDS::Listener** (p. 952) (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. 952) mechanism is used first and then the **DDS::WaitSet** (p. 1411) objects are signalled.

The statuses may be classified into:

- *read communication statuses*: i.e., those that are related to arrival of data, namely DDS::StatusKind::DATA_ON_READERS_STATUS and DDS::StatusKind::DATA_AVAILABLE_STATUS.

- *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::StatusKind::DATA_ON_READERS_STATUS

and DDS::StatusKind::DATA_AVAILABLE_STATUS) 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).

5.69.2 Changes in Status

Associated with each one of an **DDS::Entity** (p. 845)'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.

5.69.2.1 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 false each time the application accesses the plain communication status via the proper `get_<plain communication status>()` operation on the **DDS::Entity** (p. 845).

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::StatusKind::REQUESTED_DEADLINE_MISSED_STATUS will become TRUE each time new deadline occurs (which increases the **DDS::RequestedDeadlineMissedStatus::total_count** (p. 1105) field). The value changes to FALSE when the application accesses the status via the corresponding **DDS::DataReader::get_requested_deadline_missed_status** (p. 445) method on the proper **Entity** (p. 845)

5.69.2.2 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. 1409), **DDS::SampleStateKind** (p. 1161), or **DDS::InstanceStateKind** (p. 907) of any existing sample changes for any reason other than a call to **DDS::TypedDataReader::read** (p. 1341), **DDS::TypedDataReader::take** (p. 1342) 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. 907) 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. 499) that owns it if **OWNERSHIP** (p. 283) QoS kind= DDS::OwnershipQosPolicyKind::EXCLUSIVE_OWNERSHIP_QOS
    - or by any **DDS::DataWriter** (p. 499) if **OWNERSHIP** (p. 283) QoS kind= DDS::OwnershipQosPolicyKind::SHARED_OWNERSHIP_QOS
  - The loss of liveliness of the **DDS::DataWriter** (p. 499) of an instance for which there is no other **DDS::DataWriter** (p. 499).
  - The arrival of the notification that an instance has been unregistered by the only **DDS::DataWriter** (p. 499) that is known to be writing the instance.

Depending on the kind of `StatusChangedFlag`, the flag transitions to FALSE again as follows:

- The DDS::StatusKind::DATA_AVAILABLE_STATUS `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. 433).
- The DDS::StatusKind::DATA_ON_READERS_STATUS `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. 433) belonging to the **DDS::Subscriber** (p. 1201).
- The read or take operation (or their variants) is called on any **DDS::DataReader** (p. 433) belonging to the **DDS::Subscriber** (p. 1201).

See also:

DDS::Listener (p. 952)
DDS::WaitSet (p. 1411), **DDS::Condition** (p. 408)

5.69.3 Enumeration Type Documentation

5.69.3.1 enum DDS::StatusMask

A bit-mask (list) of concrete status types, i.e. DDS::StatusKind[].

The bit-mask is an efficient and compact representation of a fixed-length list of DDS::StatusKind 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.

Enumerator:

STATUS_MASK_NONE No bits are set.

STATUS_MASK_ALL All bits are set.

5.69.3.2 enum DDS::StatusKind

Type for *status* kinds.

Each concrete **DDS::Entity** (p. 845) is associated with a set of *`Status` objects whose values represent the communication status of that **DDS::Entity** (p. 845).

The communication statuses whose changes can be communicated to the application depend on the **DDS::Entity** (p. 845).

Each status value can be accessed with a corresponding method on the **DDS::Entity** (p. 845). The changes on these status values cause activation of the corresponding

DDS::StatusCondition (p. 1183) objects and trigger invocation of

the proper **DDS::Listener** (p. 952) objects to asynchronously inform the application.

See also:

DDS::Entity (p. 845), **DDS::StatusCondition** (p. 1183), **DDS::Listener** (p. 952)

Enumerator:

INCONSISTENT_TOPIC_STATUS Another topic exists with the same name but different characteristics.

Entity:

DDS::Topic (p. 1258)

Status:

DDS::InconsistentTopicStatus (p. 903)

Listener:

DDS::TopicListener (p. 1278)

OFFERED_DEADLINE_MISSED_STATUS The deadline that the **DDS::DataWriter** (p. 499) has committed through its **DDS::DeadlineQosPolicy** (p. 557) was not respected for a specific instance.

Entity:

DDS::DataWriter (p. 499)

QoS:

DEADLINE (p. 281)

Status:

DDS::OfferedDeadlineMissedStatus (p. 989)

Listener:

DDS::DataWriterListener (p. 524)

REQUESTED_DEADLINE_MISSED_STATUS The deadline that the **DDS::DataReader** (p. 433) was expecting through its **DDS::DeadlineQosPolicy** (p. 557) was not respected for a specific instance.

Entity:

DDS::DataReader (p. 433)

QoS:

DEADLINE (p. 281)

Status:

DDS::RequestedDeadlineMissedStatus (p. 1105)

Listener:

DDS::DataReaderListener (p. 461)

OFFERED_INCOMPATIBLE_QOS_STATUS A QosPolicy value was incompatible with what was requested.

Entity:

DDS::DataWriter (p. 499)

Status:

DDS::OfferedIncompatibleQosStatus (p. 991)

Listener:

DDS::DataWriterListener (p. 524)

REQUESTED_INCOMPATIBLE_QOS_STATUS A QosPolicy value was incompatible with what is offered.

Entity:

DDS::DataReader (p. 433)

Status:

DDS::RequestedIncompatibleQosStatus (p. 1107)

Listener:

DDS::DataReaderListener (p. 461)

SAMPLE_LOST_STATUS A sample has been lost (i.e. was never received).

Entity:

DDS::Subscriber (p. 1201)

Status:

DDS::SampleLostStatus (p. 1158)

Listener:

DDS::SubscriberListener (p. 1226)

SAMPLE_REJECTED_STATUS A (received) sample has been rejected.

Entity:

DDS::DataReader (p. 433)

QoS:

RESOURCE_LIMITS (p. 298)

Status:

DDS::SampleRejectedStatus (p. 1159)

Listener:

DDS::DataReaderListener (p. 461)

DATA_ON_READERS_STATUS New data is available.

Entity:

DDS::Subscriber (p. 1201)

Listener:

DDS::SubscriberListener (p. 1226)

DATA_AVAILABLE_STATUS One or more new data samples have been received.

Entity:

DDS::DataReader (p. 433)

Listener:

DDS::DataReaderListener (p. 461)

LIVELINESS_LOST_STATUS The liveliness that the **DDS::DataWriter** (p. 499) has committed to through its **DDS::LivelinessQosPolicy** (p. 960) was not respected, thus **DDS::DataReader** (p. 433) entities will consider the **DDS::DataWriter** (p. 499) as no longer alive.

Entity:

DDS::DataWriter (p. 499)

QoS:

LIVELINESS (p. 286)

Status:

DDS::LivelinessLostStatus (p. 958)

Listener:

DDS::DataWriterListener (p. 524)

LIVELINESS_CHANGED_STATUS The liveliness of one or more **DDS::DataWriter** (p. 499) that were writing instances read through the **DDS::DataReader** (p. 433) has changed. Some **DDS::DataWriter** (p. 499) have become alive or not_alive.

Entity:

DDS::DataReader (p. 433)

QoS:

LIVELINESS (p. 286)

Status:

DDS::LivelinessChangedStatus (p. 956)

Listener:

DDS::DataReaderListener (p. 461)

PUBLICATION_MATCHED_STATUS The **DDS::DataWriter** (p. 499) has found **DDS::DataReader** (p. 433) that matches the **DDS::Topic** (p. 1258) and has compatible QoS.

Entity:

DDS::DataWriter (p. 499)

Status:

DDS::PublicationMatchedStatus (p. 1041)

Listener:

DDS::DataWriterListener (p. 524)

SUBSCRIPTION_MATCHED_STATUS The **DDS::DataReader** (p. 433) has found **DDS::DataWriter** (p. 499) that matches the **DDS::Topic** (p. 1258) and has compatible QoS.

Entity:

DDS::DataReader (p. 433)

Status:

DDS::SubscriptionMatchedStatus (p. 1244)

Listener:

DDS::DataReaderListener (p. 461)

RELIABLE_WRITER_CACHE_CHANGED_STATUS

*<<**eXtension**>>* *(p. 174)* 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 unacknowledge samples), full (i.e. the sample count has reached the value specified in **DDS::ResourceLimitsQosPolicy::max_samples** (p. 1112)), or the number of samples has reached a high (see **DDS::RtpsReliableWriterProtocol_t::high_watermark** (p. 1131)) or low (see **DDS::RtpsReliableWriterProtocol_t::low_watermark** (p. 1130)) watermark.

Entity:

DDS::DataWriter (p. 499)

Status:

DDS::ReliableWriterCacheChangedStatus (p. 1101)

Listener:

DDS::DataWriterListener (p. 524)

RELIABLE_READER_ACTIVITY_CHANGED_STATUS *<<eXtension>>* *(p. 174)* 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. 1128) and **DDS::ReliableReaderActivityChangedStatus** (p. 1098).

See also:

DDS::RtpsReliableWriterProtocol_t (p. 1128)
DDS::ReliableReaderActivityChangedStatus (p. 1098)

DATA_WRITER_CACHE_STATUS *<<eXtension>>* *(p. 174)* The status of the writer's cache.

DATA_WRITER_PROTOCOL_STATUS *<<eXtension>>* *(p. 174)* The status of a writer's internal protocol related metrics

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_READER_CACHE_STATUS *<<eXtension>>* *(p. 174)* The status of the reader's cache.

DATA_READER_PROTOCOL_STATUS *<<eXtension>>* *(p. 174)* 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.

5.70 Exception Codes

<<*eXtension*>> *(p. 174)* **Exception** (p. 861) codes.

Enumerations

ˆ enum **DDS::ExceptionCode_t** {
DDS::DDS_NO_EXCEPTION_CODE,
DDS::DDS_USER_EXCEPTION_CODE,
DDS::DDS_SYSTEM_EXCEPTION_CODE,
DDS::DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE,
DDS::DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE,
DDS::DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE,
DDS::DDS_BADKIND_USER_EXCEPTION_CODE,
DDS::DDS_BOUNDS_USER_EXCEPTION_CODE,
DDS::DDS_IMMUTABLE_TYPECODE_SYSTEM_-EXCEPTION_CODE = 8,
DDS::DDS_BAD_MEMBER_NAME_USER_EXCEPTION_-CODE = 9,
DDS::DDS_BAD_MEMBER_ID_USER_EXCEPTION_CODE = 10 }
Exceptions used by the ***DDS::TypeCode*** *(p. 1301) class.*

5.70.1 Detailed Description

<<*eXtension*>> *(p. 174)* **Exception** (p. 861) codes.

These exceptions are used for error handling by the **Type Code Support** (p. 55) API.

5.70.2 Enumeration Type Documentation

5.70.2.1 enum DDS::ExceptionCode_t

Exceptions used by the **DDS::TypeCode** (p. 1301) class.

Enumerator:

DDS_NO_EXCEPTION_CODE No failure occurred.

DDS_USER_EXCEPTION_CODE User exception.

This class is based on a similar class in CORBA.

DDS_SYSTEM_EXCEPTION_CODE System exception.

This class is based on a similar class in CORBA.

DDS_BAD_PARAM_SYSTEM_EXCEPTION_CODE **Exception** (p. 861) thrown when a parameter passed to a call is considered illegal.

DDS_NO_MEMORY_SYSTEM_EXCEPTION_CODE **Exception** (p. 861) thrown when there is not enough memory for a dynamic memory allocation.

DDS_BAD_TYPECODE_SYSTEM_EXCEPTION_CODE **Exception** (p. 861) 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** (p. 1301) 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. 1301) that was received from a remote object.

The built-in publication and subscription readers provide access to information about the remote **DDS::DataWriter** (p. 499) and **DDS::DataReader** (p. 433) entities in the distributed system. Among other things, the data from these built-in readers contains the **DDS::TypeCode** (p. 1301) for these entities. Modifying this received **DDS::TypeCode** (p. 1301) is not permitted.

DDS_BAD_MEMBER_NAME_USER_EXCEPTION_CODE The specified **DDS::TypeCode** (p. 1301) 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::ExceptionCode_t::BAD_MEMBER_ID_USER_-EXCEPTION_CODE

DDS_BAD_MEMBER_ID_USER_EXCEPTION_CODE The specified **DDS::TypeCode** (p. 1301) 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::ExceptionCode_t::BAD_MEMBER_NAME_USER_-EXCEPTION_CODE

5.71 Time Support

Time and duration types and defines.

Classes

- struct **DDS::Time_t**

  Type for time *representation.*

- struct **DDS::Duration_t**

  Type for duration *representation.*

Functions

- System::Boolean **DDS::Time_t::is_zero** ()

  Check if time is zero.

- System::Boolean **DDS::Time_t::is_invalid_time** ()
- System::Boolean **DDS::Duration_t::is_infinite** ()
- System::Boolean **DDS::Duration_t::is_zero** ()

Properties

- static System::Int32 **DDS::Time_t::TIME_INVALID_SEC** `[get]`

  A sentinel indicating an invalid second of time.

- static System::Int32 **DDS::Time_t::TIME_INVALID_NSEC** `[get]`

  A sentinel indicating an invalid nano-second of time.

- static **Time_t DDS::Time_t::TIME_ZERO** `[get]`

  The default instant in time: zero seconds and zero nanoseconds.

- static **Time_t DDS::Time_t::TIME_INVALID** `[get]`

  A sentinel indicating an invalid time.

- static System::Int32 **DDS::Duration_t::DURATION_ZERO_SEC** `[get]`

  A zero-length second period of time.

- static System::Int32 **DDS::Duration_t::DURATION_ZERO_NSEC** [get]

  A zero-length nano-second period of time.

- static System::Int32 **DDS::Duration_t::DURATION_INFINITE_SEC** [get]

  An infinite second period of time.

- static System::Int32 **DDS::Duration_t::DURATION_INFINITE_NSEC** [get]

  An infinite nano-second period of time.

- static **Duration_t DDS::Duration_t::DURATION_INFINITE** [get]

  An infinite period of time.

- static **Duration_t DDS::Duration_t::DURATION_ZERO** [get]

  A zero-length period of time.

5.71.1 Detailed Description

Time and duration types and defines.

5.71.2 Function Documentation

5.71.2.1 System::Boolean DDS::Time_t::is_zero () [inline, inherited]

Check if time is zero.

Returns:

true if the given time is equal to **DDS::Time_t::TIME_ZERO** (p. 252) or false otherwise.

5.71.2.2 System::Boolean DDS::Time_t::is_invalid_time () [inline, inherited]

Returns:

true if the given time is not valid (i.e. is negative)

5.71.2.3 System::Boolean DDS::Duration_t::is_infinite () [inline, inherited]

Returns:

true if the given duration is of infinite length.

5.71.2.4 System::Boolean DDS::Duration_t::is_zero () [inline, inherited]

Returns:

true if the given duration is of zero length.

5.71.3 Properties

5.71.3.1 System:: Int32 DDS::Time_t::TIME_INVALID_SEC [static, get, inherited]

A sentinel indicating an invalid second of time.

5.71.3.2 System:: Int32 DDS::Time_t::TIME_INVALID_NSEC [static, get, inherited]

A sentinel indicating an invalid nano-second of time.

5.71.3.3 Time_t DDS::Time_t::TIME_ZERO [static, get, inherited]

The default instant in time: zero seconds and zero nanoseconds.

5.71.3.4 Time_t DDS::Time_t::TIME_INVALID [static, get, inherited]

A sentinel indicating an invalid time.

5.71.3.5 System:: Int32 DDS::Duration_t::DURATION_ZERO_SEC [static, get, inherited]

A zero-length second period of time.

5.71.3.6 System:: Int32 DDS::Duration_t::DURATION_ZERO_NSEC [static, get, inherited]

A zero-length nano-second period of time.

5.71.3.7 System:: Int32 DDS::Duration_t::DURATION_INFINITE_SEC [static, get, inherited]

An infinite second period of time.

5.71.3.8 System:: Int32 DDS::Duration_t::DURATION_INFINITE_NSEC [static, get, inherited]

An infinite nano-second period of time.

5.71.3.9 Duration_t DDS::Duration_t::DURATION_INFINITE [static, get, inherited]

An infinite period of time.

5.71.3.10 Duration_t DDS::Duration_t::DURATION_ZERO [static, get, inherited]

A zero-length period of time.

5.72 GUID Support

<<*eXtension*>> *(p. 174)* GUID type and defines.

Classes

- struct **DDS::GUID_t**

  Type for GUID *(Global Unique Identifier) representation.*

Properties

- static **GUID_t DDS::GUID_t::GUID_UNKNOWN** `[get]`

  Unknown GUID.

- static **GUID_t DDS::GUID_t::GUID_AUTO** `[get]`

  Indicates that RTI Data Distribution Service should choose an appropriate virtual GUID.

5.72.1 Detailed Description

<<*eXtension*>> *(p. 174)* GUID type and defines.

5.72.2 Properties

5.72.2.1 GUID_t DDS::GUID_t::GUID_UNKNOWN [static, get, inherited]

Unknown GUID.

5.72.2.2 GUID_t DDS::GUID_t::GUID_AUTO [static, get, inherited]

Indicates that RTI Data Distribution Service should choose an appropriate virtual GUID.

If this special value is assigned to **DDS::DataWriterProtocolQosPolicy::virtual_guid** (p. 530) or **DDS::DataReaderProtocolQosPolicy::virtual_guid** (p. 467), RTI Data Distribution Service will assign the virtual GUID automatically based on the RTPS or physical GUID.

5.73 Sequence Number Support

<<*eXtension*>> *(p. 174)* **Sequence** (p. 1163) number type and defines.

Classes

- struct **DDS::SequenceNumber_t**

  Type for sequence *number representation.*

Functions

- static Int32 **DDS::SequenceNumber_t::sequence_number_compare** (**SequenceNumber_t**^ sn1, **SequenceNumber_t**^ sn2)

  Compares two sequence numbers.

Properties

- static **SequenceNumber_t** **DDS::SequenceNumber_t::SEQUENCE_NUMBER_UNKNOWN** `[get]`

  Unknown sequence number.

- static **SequenceNumber_t** **DDS::SequenceNumber_t::SEQUENCE_NUMBER_ZERO** `[get]`

  Zero value for the sequence number.

- static **SequenceNumber_t** **DDS::SequenceNumber_t::SEQUENCE_NUMBER_MAX** `[get]`

  Highest, most positive value for the sequence number.

5.73.1 Detailed Description

<<*eXtension*>> *(p. 174)* **Sequence** (p. 1163) number type and defines.

5.73.2 Function Documentation

5.73.2.1 static Int32 DDS::SequenceNumber_t::sequence_number_compare (SequenceNumber_t^ *sn1*, SequenceNumber_t^ *sn2*) [inline, static, inherited]

Compares two sequence numbers.

Parameters:

sn1 <<*in*>> *(p. 175)* **Sequence** (p. 1163) number to compare. Cannot be null.

sn2 <<*in*>> *(p. 175)* **Sequence** (p. 1163) 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.

5.73.3 Properties

5.73.3.1 SequenceNumber_t DDS::SequenceNumber_t::SEQUENCE_NUMBER_UNKNOWN [static, get, inherited]

Unknown sequence number.

5.73.3.2 SequenceNumber_t DDS::SequenceNumber_t::SEQUENCE_NUMBER_ZERO [static, get, inherited]

Zero value for the sequence number.

5.73.3.3 SequenceNumber_t DDS::SequenceNumber_t::SEQUENCE_NUMBER_MAX [static, get, inherited]

Highest, most positive value for the sequence number.

5.74 Thread Settings

The properties of a thread of execution.

Classes

- class **DDS::ThreadSettings_t**

  The properties of a thread of execution.

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_KIND_MASK_DEFAULT }

  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. 1250) *affects processor affinity for thread-related QoS policies that apply to multiple threads.*

5.74.1 Detailed Description

The properties of a thread of execution.

5.74.2 Enumeration Type Documentation

5.74.2.1 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.

See also:

DDS::ThreadSettingsKindMask

Enumerator:

THREAD_SETTINGS_FLOATING_POINT Code executed within the thread may perform floating point operations.

THREAD_SETTINGS_STDIO Code executed within the thread may access standard I/O.

THREAD_SETTINGS_REALTIME_PRIORITY The thread will be schedule on a real-time basis.

THREAD_SETTINGS_PRIORITY_ENFORCE Strictly enforce this thread's priority.

THREAD_SETTINGS_KIND_MASK_DEFAULT The mask of default thread options.

5.74.2.2 enum DDS::ThreadSettingsCpuRotationKind

Determines how **DDS::ThreadSettings_t::cpu_list** (p. 1250) affects processor affinity for thread-related QoS policies that apply to multiple threads.

5.74.3 Controlling CPU Core Affinity for RTI Threads

Most thread-related QoS settings apply to a single thread (such as for the **DDS::EventQosPolicy** (p. 858), **DDS::DatabaseQosPolicy** (p. 428), and **DDS::AsynchronousPublisherQosPolicy** (p. 371)). However, the thread settings in the **DDS::ReceiverPoolQosPolicy** (p. 1090) 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.

If **DDS::ThreadSettings_t::cpu_list** (p. 1250) is empty, the rotation is irrelevant since no affinity adjustment will occur. Suppose instead that **DDS::ThreadSettings_t::cpu_list** (p. 1250) = {0, 1} and that the middleware creates three receive threads: {A, B, C}. If **DDS::ThreadSettings_t::cpu_rotation** (p. 1250) is DDS::ThreadSettingsCpuRotationKind::THREAD_SETTINGS_CPU_NO_ROTATION, 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. 1250) is DDS::ThreadSettingsCpuRotationKind::THREAD_SETTINGS_CPU_RR_ROTATION, each thread will be assigned in round-robin fashion to one of the

processors in **DDS::ThreadSettings_t::cpu_list** (p. 1250); 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.

Enumerator:

THREAD_SETTINGS_CPU_NO_ROTATION Any thread controlled by this QoS can run on any listed processor, as determined by OS scheduling.

THREAD_SETTINGS_CPU_RR_ROTATION Threads controlled by this QoS will be assigned one processor from the list in round-robin order.

5.75 QoS Policies

Quality of Service (QoS) policies.

Modules

- **USER_DATA**

  Attaches a buffer of opaque data that is distributed by means of ***Built-in Topics*** *(p. 42) during discovery.*

- **TOPIC_DATA**

  Attaches a buffer of opaque data that is distributed by means of ***Built-in Topics*** *(p. 42) during discovery.*

- **GROUP_DATA**

  Attaches a buffer of opaque data that is distributed by means of ***Built-in Topics*** *(p. 42) during discovery.*

- **DURABILITY**

  This QoS policy specifies whether or not RTI Data Distribution Service will store and deliver previously published data samples to new ***DDS::DataReader*** *(p. 433) entities that join the network later.*

- **PRESENTATION**

  Specifies how the samples representing changes to data instances are presented to a subscribing application.

- **DEADLINE**

  Expresses the maximum duration (deadline) within which an instance is expected to be updated.

- **LATENCY_BUDGET**

  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.

- **OWNERSHIP**

  Specifies whether it is allowed for multiple ***DDS::DataWriter*** *(p. 499) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.*

- **OWNERSHIP_STRENGTH**

  Specifies the value of the strength used to arbitrate among multiple ***DDS::DataWriter*** *(p. 499) objects that attempt to modify the same instance of a data type (identified by* ***DDS::Topic*** *(p. 1258) + key).*

- **LIVELINESS**

  Specifies and configures the mechanism that allows ***DDS::DataReader*** *(p. 433) entities to detect when* ***DDS::DataWriter*** *(p. 499) entities become disconnected or "dead.".*

- **TIME_BASED_FILTER**

  Filter that allows a ***DDS::DataReader*** *(p. 433) to specify that it is interested only in (potentially) a subset of the values of the data.*

- **PARTITION**

  Set of strings that introduces a logical partition among the topics visible by a ***DDS::Publisher*** *(p. 1044) and a* ***DDS::Subscriber*** *(p. 1201).*

- **RELIABILITY**

  Indicates the level of reliability offered/requested by RTI Data Distribution Service.

- **DESTINATION_ORDER**

  Controls the criteria used to determine the logical order among changes made by ***DDS::Publisher*** *(p. 1044) entities to the same instance of data (i.e., matching* ***DDS::Topic*** *(p. 1258) and key).*

- **HISTORY**

  Specifies the behavior of RTI Data Distribution Service 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.

- **DURABILITY_SERVICE**

  Various settings to configure the external RTI Persistence Service *used by RTI Data Distribution Service for DataWriters with a* ***DDS::DurabilityQosPolicy*** *(p. 709) setting of DDS::DurabilityQosPolicyKind::PERSISTENT_DURABILITY_QOS or DDS::DurabilityQosPolicyKind::TRANSIENT_DURABILITY_QOS.*

- **RESOURCE_LIMITS**

  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.

- **TRANSPORT_PRIORITY**

  This QoS policy allows the application to take advantage of transports that are capable of sending messages with different priorities.

- **LIFESPAN**

*Specifies how long the data written by the **DDS::DataWriter** (p. 499) is considered valid.*

- **WRITER_DATA_LIFECYCLE**

  *Controls how a **DataWriter** (p. 499) handles the lifecycle of the instances (keys) that it is registered to manage.*

- **READER_DATA_LIFECYCLE**

  *Controls how a **DataReader** (p. 433) manages the lifecycle of the data that it has received.*

- **ENTITY_FACTORY**

  *A QoS policy for all **DDS::Entity** (p. 845) types that can act as factories for one or more other **DDS::Entity** (p. 845) types.*

- **Extended Qos Support**

  <<**eXtension**>> (p. 174) *Types and defines used in extended QoS policies.*

- **TRANSPORT_SELECTION**

  <<**eXtension**>> (p. 174) *Specifies the physical transports a **DDS::DataWriter** (p. 499) or **DDS::DataReader** (p. 433) may use to send or receive data.*

- **TRANSPORT_UNICAST**

  <<**eXtension**>> (p. 174) *Specifies a subset of transports and a port number that can be used by an **Entity** (p. 845) to receive data.*

- **TRANSPORT_MULTICAST**

  <<**eXtension**>> (p. 174) *Specifies the multicast address on which a **DDS::DataReader** (p. 433) 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. 577) level) transports with which to receive the multicast data.*

- **DISCOVERY**

  <<**eXtension**>> (p. 174) *Specifies the attributes required to discover participants in the domain.*

- **TRANSPORT_BUILTIN**

  <<**eXtension**>> (p. 174) *Specifies which built-in transports are used.*

- **WIRE_PROTOCOL**

  <<**eXtension**>> (p. 174) *Specifies the wire protocol related attributes for the **DDS::DomainParticipant** (p. 577).*

- **DATA_READER_RESOURCE_LIMITS**

  <<**eXtension**>> (p. 174) *Various settings that configure how DataReaders allocate and use physical memory for internal resources.*

- **DATA_WRITER_RESOURCE_LIMITS**

  <<**eXtension**>> (p. 174) *Various settings that configure how a* ***DDS::DataWriter*** *(p. 499) allocates and uses physical memory for internal resources.*

- **DATA_READER_PROTOCOL**

  <<**eXtension**>> (p. 174) *Specifies the DataReader-specific protocol QoS.*

- **DATA_WRITER_PROTOCOL**

  <<**eXtension**>> (p. 174) *Along with* ***DDS::WireProtocolQosPolicy*** *(p. 1423) and* ***DDS::DataReaderProtocolQosPolicy*** *(p. 465), this QoS policy configures the DDS on-the-network protocol (RTPS).*

- **SYSTEM_RESOURCE_LIMITS**

  <<**eXtension**>> (p. 174) *Configures DomainParticipant-independent resources used by RTI Data Distribution Service.*

- **DOMAIN_PARTICIPANT_RESOURCE_LIMITS**

  <<**eXtension**>> (p. 174) *Various settings that configure how a* ***DDS::DomainParticipant*** *(p. 577) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.*

- **EVENT**

  <<**eXtension**>> (p. 174) *Configures the internal thread in a* ***DomainParticipant*** *(p. 577) that handles timed events.*

- **DATABASE**

  <<**eXtension**>> (p. 174) *Various threads and resource limits settings used by RTI Data Distribution Service to control its internal database.*

- **RECEIVER_POOL**

  <<**eXtension**>> (p. 174) *Configures threads used by RTI Data Distribution Service to receive and process data from transports (for example, UDP sockets).*

- **PUBLISH_MODE**

  <<**eXtension**>> (p. 174) *Specifies how RTI Data Distribution Service sends application data on the network. This QoS policy can be used to tell RTI Data Distribution Service to use its* own *thread to send data, instead of the user thread.*

^ DISCOVERY_CONFIG

<<**eXtension**>> (p. 174) *Specifies the discovery configuration QoS.*

^ TYPESUPPORT

<<**eXtension**>> (p. 174) *Allows you to attach application-specific values to a* ***DataWriter*** *(p. 499) or* ***DataReader*** *(p. 433) that are passed to the serialization or deserialization routine of the associated data type.*

^ ASYNCHRONOUS_PUBLISHER

<<**eXtension**>> (p. 174) *Specifies the asynchronous publishing settings of the* ***DDS::Publisher*** *(p. 1044) instances.*

^ EXCLUSIVE_AREA

<<**eXtension**>> (p. 174) *Configures multi-thread concurrency and deadlock prevention capabilities.*

^ BATCH

<<**eXtension**>> (p. 174) *Batch QoS policy used to enable batching in* ***DDS::DataWriter*** *(p. 499) instances.*

^ LOCATORFILTER

<<**eXtension**>> (p. 174) *The QoS policy used to report the configuration of a MultiChannel* ***DataWriter*** *(p. 499) as part of* ***DDS::PublicationBuiltinTopicData*** *(p. 1030).*

^ MULTICHANNEL

<<**eXtension**>> (p. 174) *Configures the ability of a* ***DataWriter*** *(p. 499) to send data on different multicast groups (addresses) based on the value of the data.*

^ PROPERTY

<<**eXtension**>> (p. 174) *Stores name/value (string) pairs that can be used to configure certain parameters of RTI Data Distribution Service 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_NAME

<<**eXtension**>> (p. 174) *Assigns a name to a* ***DDS::DomainParticipant*** *(p. 577). This name will be visible during the discovery process and in RTI tools to help you visualize and debug your system.*

^ PROFILE

<<**eXtension**>> (p. 174) *Configures the way that XML documents containing QoS profiles are loaded by RTI Data Distribution Service.*

Classes

ˆ struct **DDS::QosPolicyCount**

Type to hold a counter for a DDS::QosPolicyId_t.

ˆ class **DDS::QosPolicyCountSeq**

Declares IDL `sequence` < ***DDS::QosPolicyCount*** *(p. 1080)* >.

Enumerations

ˆ 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::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::EXCLUSIVEAREA_QOS_POLICY_ID ,

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::DDS_BATCH_QOS_POLICY_ID,

DDS::DDS_PROFILE_QOS_POLICY_ID,

DDS::DDS_LOCATORFILTER_QOS_POLICY_ID,

DDS::DDS_MULTICHANNEL_QOS_POLICY_ID }

Type to identify `QosPolicies`.

5.75.1 Detailed Description

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).

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. 845) objects in the system such as **DDS::Topic** (p. 1258), **DDS::DataWriter** (p. 499), **DDS::DataReader** (p. 433), **DDS::Publisher** (p. 1044), **DDS::Subscriber** (p. 1201), and **DDS::DomainParticipant** (p. 577).

5.75.2 Specifying QoS on entities

QosPolicies can be set programmatically when an **DDS::Entity** (p. 845) is created, or modified with the **DDS::Entity** (p. 845)'s **set_qos (abstract)** (p. 846) method.

QosPolicies can also be configured from XML resources (files, strings). With this approach, you can change the QoS without recompiling the application. For more information, see **Configuring QoS Profiles with XML** (p. 135).

To customize a **DDS::Entity** (p. 845)'s QoS before creating the entity, the correct pattern is:

- First, initialize a QoS object with the appropriate INITIALIZER constructor.
- Call the relevant get_<entity>_default_qos() method.
- Modify the QoS values as desired.
- Finally, create the entity.

Each `QosPolicy` is treated independently from the others. This approach has the advantage of being very extensible. However, there may be cases where several policies are in conflict. Consistency checking is performed each time the

policies are modified via the **set_qos (abstract)** (p. 846) operation, or when the **DDS::Entity** (p. 845) 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 Data Distribution Service 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. 845) creation time or else prior to calling the **DDS::Entity::enable** (p. 848) operation on the entity.

Each **DDS::Entity** (p. 845) can be configured with a list of `QosPolicy` objects. However, not all QosPolicies are supported by each **DDS::Entity** (p. 845). For instance, a **DDS::DomainParticipant** (p. 577) supports a different set of QosPolicies than a **DDS::Topic** (p. 1258) or a **DDS::Publisher** (p. 1044).

5.75.3 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. 1201) requests to receive data reliably while the corresponding **DDS::Publisher** (p. 1044) defines a best-effort policy, communication will not happen as requested.

To address this issue and maintain the desirable decoupling of publication and subscription as much as possible, the QosPolicy specification follows the **subscriber-requested, publisher-offered pattern**.

In this pattern, the subscriber side can specify a "requested" value for a particular `QosPolicy`. The publisher side specifes an "offered" value for that `QosPolicy`. RTI Data Distribution Service 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 Data Distribution Service will not establish communications between the two **DDS::Entity** (p. 845) objects and will record this fact by means of the DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS on the publisher end and DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS on the subscriber end. The application can detect this fact by means of a **DDS::Listener** (p. 952) or a **DDS::Condition** (p. 408).

The following **properties** are defined on a QosPolicy.

- **RxO (p. 268) 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. 268) property:

  - **RxO** (p. 268) = **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. 268) = **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. 268) = **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.

ˆ **Changeable** (p. **269**) **property**

Determines whether a QosPolicy can be changed.

NO (p. **269**) – policy can only be specified at **DDS::Entity** (p. 845) creation time.

UNTIL ENABLE (p. **269**) – policy can only be changed before the **DDS::Entity** (p. 845) is enabled.

YES (p. **269**) – policy can be changed at any time.

5.75.4 Enumeration Type Documentation

5.75.4.1 enum DDS::QosPolicyId_t

Type to identify `QosPolicies`.

Enumerator:

INVALID_QOS_POLICY_ID Identifier for an invalid QoS policy.

USERDATA_QOS_POLICY_ID Identifier for **DDS::UserDataQosPolicy** (p. 1403).

DURABILITY_QOS_POLICY_ID Identifier for **DDS::DurabilityQosPolicy** (p. 709).

PRESENTATION_QOS_POLICY_ID Identifier for **DDS::PresentationQosPolicy** (p. 1012).

DEADLINE_QOS_POLICY_ID Identifier for **DDS::DeadlineQosPolicy** (p. 557).

LATENCYBUDGET_QOS_POLICY_ID Identifier for **DDS::LatencyBudgetQosPolicy** (p. 948).

OWNERSHIP_QOS_POLICY_ID Identifier for **DDS::OwnershipQosPolicy** (p. 993).

OWNERSHIPSTRENGTH_QOS_POLICY_ID Identifier for **DDS::OwnershipStrengthQosPolicy** (p. 1000).

LIVELINESS_QOS_POLICY_ID Identifier for **DDS::LivelinessQosPolicy** (p. 960).

TIMEBASEDFILTER_QOS_POLICY_ID Identifier for **DDS::TimeBasedFilterQosPolicy** (p. 1254).

PARTITION_QOS_POLICY_ID Identifier for **DDS::PartitionQosPolicy** (p. 1008).

RELIABILITY_QOS_POLICY_ID Identifier for **DDS::ReliabilityQosPolicy** (p. 1094).

DESTINATIONORDER_QOS_POLICY_ID Identifier for **DDS::DestinationOrderQosPolicy** (p. 560).

HISTORY_QOS_POLICY_ID Identifier for **DDS::HistoryQosPolicy** (p. 898).

RESOURCELIMITS_QOS_POLICY_ID Identifier for **DDS::ResourceLimitsQosPolicy** (p. 1109).

ENTITYFACTORY_QOS_POLICY_ID Identifier for **DDS::EntityFactoryQosPolicy** (p. 851).

WRITERDATALIFECYCLE_QOS_POLICY_ID Identifier for **DDS::WriterDataLifecycleQosPolicy** (p. 1431).

READERDATALIFECYCLE_QOS_POLICY_ID Identifier for **DDS::ReaderDataLifecycleQosPolicy** (p. 1087).

TOPICDATA_QOS_POLICY_ID Identifier for **DDS::TopicDataQosPolicy** (p. 1276).

GROUPDATA_QOS_POLICY_ID Identifier for **DDS::GroupDataQosPolicy** (p. 890).

TRANSPORTPRIORITY_QOS_POLICY_ID Identifier for **DDS::TransportPriorityQosPolicy** (p. 1292).

LIFESPAN_QOS_POLICY_ID Identifier for **DDS::LifespanQosPolicy** (p. 950).

DURABILITYSERVICE_QOS_POLICY_ID Identifier for **DDS::DurabilityServiceQosPolicy** (p. 714).

WIREPROTOCOL_QOS_POLICY_ID *<<eXtension>> (p. 174)* Identifier for **DDS::WireProtocolQosPolicy** (p. 1423)

DISCOVERY_QOS_POLICY_ID *<<eXtension>> (p. 174)* Identifier for **DDS::DiscoveryQosPolicy** (p. 571)

DATAREADERRESOURCELIMITS_QOS_POLICY_ID *<<eXtension>> (p. 174)* Identifier for **DDS::DataReaderResourceLimitsQosPolicy** (p. 486)

DATAWRITERRESOURCELIMITS_QOS_POLICY_ID *<<eXtension>> (p. 174)* Identifier for **DDS::DataWriterResourceLimitsQosPolicy** (p. 552)

DATAREADERPROTOCOL_QOS_POLICY_ID
<<eXtension>> *(p. 174)* Identifier for **DDS::DataReaderProtocolQosPolicy** (p. 465)

DATAWRITERPROTOCOL_QOS_POLICY_ID
<<eXtension>> *(p. 174)* Identifier for **DDS::DataWriterProtocolQosPolicy** (p. 529)

DOMAINPARTICIPANTRESOURCELIMITS_QOS_POLICY_ID
<<eXtension>> *(p. 174)* Identifier for **DDS::DomainParticipantResourceLimitsQosPolicy** (p. 688)

EVENT_QOS_POLICY_ID *<<eXtension>>* *(p. 174)* Identifier for **DDS::EventQosPolicy** (p. 858)

DATABASE_QOS_POLICY_ID *<<eXtension>>* *(p. 174)* Identifier for **DDS::DatabaseQosPolicy** (p. 428)

RECEIVERPOOL_QOS_POLICY_ID *<<eXtension>>* *(p. 174)* Identifier for **DDS::ReceiverPoolQosPolicy** (p. 1090)

DISCOVERYCONFIG_QOS_POLICY_ID *<<eXtension>>* *(p. 174)* Identifier for **DDS::DiscoveryConfigQosPolicy** (p. 563)

EXCLUSIVEAREA_QOS_POLICY_ID *<<eXtension>>* *(p. 174)* Identifier for **DDS::ExclusiveAreaQosPolicy** (p. 862)

SYSTEMRESOURCELIMITS_QOS_POLICY_ID
<<eXtension>> *(p. 174)* Identifier for **DDS::SystemResourceLimitsQosPolicy** (p. 1247)

TRANSPORTSELECTION_QOS_POLICY_ID *<<eXtension>>* *(p. 174)* Identifier for **DDS::TransportSelectionQosPolicy** (p. 1294)

TRANSPORTUNICAST_QOS_POLICY_ID *<<eXtension>>* *(p. 174)* Identifier for **DDS::TransportUnicastQosPolicy** (p. 1296)

TRANSPORTMULTICAST_QOS_POLICY_ID
<<eXtension>> *(p. 174)* Identifier for **DDS::TransportMulticastQosPolicy** (p. 1287)

TRANSPORTBUILTIN_QOS_POLICY_ID *<<eXtension>>* *(p. 174)* Identifier for **DDS::TransportBuiltinQosPolicy** (p. 1285)

TYPESUPPORT_QOS_POLICY_ID *<<eXtension>>* *(p. 174)* Identifier for **DDS::TypeSupportQosPolicy** (p. 1386)

PROPERTY_QOS_POLICY_ID *<<eXtension>>* *(p. 174)* Identifier for **DDS::PropertyQosPolicy** (p. 1023)

PUBLISHMODE_QOS_POLICY_ID *<<eXtension>>* *(p. 174)* Identifier for **DDS::PublishModeQosPolicy** (p. 1077)

ASYNCHRONOUSPUBLISHER_QOS_POLICY_ID *<<eXtension>>* *(p. 174)* Identifier for **DDS::AsynchronousPublisherQosPolicy** (p. 371)

ENTITYNAME_QOS_POLICY_ID *<<eXtension>>* *(p. 174)* Identifier for **DDS::EntityNameQosPolicy** (p. 854)

DDS_BATCH_QOS_POLICY_ID *<<eXtension>>* *(p. 174)* Identifier for **DDS::BatchQosPolicy** (p. 376)

DDS_PROFILE_QOS_POLICY_ID *<<eXtension>>* *(p. 174)* Identifier for **DDS::ProfileQosPolicy** (p. 1019)

DDS_LOCATORFILTER_QOS_POLICY_ID *<<eXtension>>* *(p. 174)* Identifier for **DDS::LocatorFilterQosPolicy** (p. 972)

DDS_MULTICHANNEL_QOS_POLICY_ID *<<eXtension>>* *(p. 174)* Identifier for **DDS::MultiChannelQosPolicy** (p. 981)

5.76 USER_DATA

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 42) during discovery.

Classes

- ^ class **DDS::UserDataQosPolicy**

  *Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 42) during discovery.*

Functions

- ^ static System::String^ **DDS::UserDataQosPolicy::get_userdata_qos_policy_name** ()

  *Stringified human-readable name for **DDS::UserDataQosPolicy** (p. 1403).*

5.76.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 42) during discovery.

5.76.2 Function Documentation

5.76.2.1 static System::String ^ DDS::UserDataQosPolicy::get_userdata_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::UserDataQosPolicy** (p. 1403).

5.77 TOPIC_DATA

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 42) during discovery.

Classes

- class **DDS::TopicDataQosPolicy**

  *Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 42) during discovery.*

Functions

- static System::String^ **DDS::TopicDataQosPolicy::get_topicdata_qos_policy_name** ()

  *Stringified human-readable name for **DDS::TopicDataQosPolicy** (p. 1276).*

5.77.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 42) during discovery.

5.77.2 Function Documentation

5.77.2.1 static System::String ^ DDS::TopicDataQosPolicy::get_topicdata_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::TopicDataQosPolicy** (p. 1276).

5.78 GROUP_DATA

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 42) during discovery.

Classes

- class **DDS::GroupDataQosPolicy**

  *Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 42) during discovery.*

Functions

- static System::String^ **DDS::GroupDataQosPolicy::get_groupdata_qos_policy_name** ()

  *Stringified human-readable name for **DDS::GroupDataQosPolicy** (p. 890).*

5.78.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 42) during discovery.

5.78.2 Function Documentation

5.78.2.1 static System::String ^ DDS::GroupDataQosPolicy::get_groupdata_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::GroupDataQosPolicy** (p. 890).

5.79 DURABILITY

This QoS policy specifies whether or not RTI Data Distribution Service will store and deliver previously published data samples to new **DDS::DataReader** (p. 433) entities that join the network later.

Classes

^ struct **DDS::DurabilityQosPolicy**

This QoS policy specifies whether or not RTI Data Distribution Service will store and deliver previously published data samples to new ***DDS::DataReader*** *(p. 433) entities that join the network later.*

Enumerations

^ enum **DDS::DurabilityQosPolicyKind** {

DDS::VOLATILE_DURABILITY_QOS,

DDS::TRANSIENT_LOCAL_DURABILITY_QOS,

DDS::TRANSIENT_DURABILITY_QOS,

DDS::PERSISTENT_DURABILITY_QOS }

Kinds of durability.

Functions

^ static System::String^ **DDS::DurabilityQosPolicy::get_durability_qos_policy_name** ()

Stringified human-readable name for ***DDS::DurabilityQosPolicy*** *(p. 709).*

5.79.1 Detailed Description

This QoS policy specifies whether or not RTI Data Distribution Service will store and deliver previously published data samples to new **DDS::DataReader** (p. 433) entities that join the network later.

5.79.2 Enumeration Type Documentation

5.79.2.1 enum DDS::DurabilityQosPolicyKind

Kinds of durability.

QoS:

DDS::DurabilityQosPolicy (p. 709)

Enumerator:

VOLATILE_DURABILITY_QOS [**default**] RTI Data Distribution Service does not need to keep any samples of data instances on behalf of any **DDS::DataReader** (p. 433) that is unknown by the **DDS::DataWriter** (p. 499) at the time the instance is written.

In other words, RTI Data Distribution Service will only attempt to provide the data to existing subscribers.

TRANSIENT_LOCAL_DURABILITY_QOS RTI Data Distribution Service will attempt to keep some samples so that they can be delivered to any potential late-joining **DDS::DataReader** (p. 433).

Which particular samples are kept depends on other QoS such as **DDS::HistoryQosPolicy** (p. 898) and **DDS::ResourceLimitsQosPolicy** (p. 1109). RTI Data Distribution Service is only required to keep the data in memory of the **DDS::DataWriter** (p. 499) that wrote the data.

Data is not required to survive the **DDS::DataWriter** (p. 499).

For this setting to be effective, you must also set the **DDS::ReliabilityQosPolicy::kind** (p. 1097) to DDS::ReliabilityQosPolicyKind::RELIABLE_RELIABILITY_QOS.

TRANSIENT_DURABILITY_QOS RTI Data Distribution Service will attempt to keep some samples so that they can be delivered to any potential late-joining **DDS::DataReader** (p. 433).

Which particular samples are kept depends on other QoS such as **DDS::HistoryQosPolicy** (p. 898) and **DDS::ResourceLimitsQosPolicy** (p. 1109). RTI Data Distribution Service 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. 499).

Data will survive the **DDS::DataWriter** (p. 499).

PERSISTENT_DURABILITY_QOS Data is kept on permanent storage, so that they can outlive a system session.

5.79.3 Function Documentation

5.79.3.1 static System::String ^ DDS::DurabilityQosPolicy::get_durability_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::DurabilityQosPolicy** (p. 709).

5.80 PRESENTATION

Specifies how the samples representing changes to data instances are presented to a subscribing application.

Classes

- struct **DDS::PresentationQosPolicy**

  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 }

  Kinds of presentation "access scope".

Functions

- static System::String^ **DDS::PresentationQosPolicy::get_presentation_qos_policy_name** ()

  *Stringified human-readable name for **DDS::PresentationQosPolicy*** *(p. 1012).*

5.80.1 Detailed Description

Specifies how the samples representing changes to data instances are presented to a subscribing application.

5.80.2 Enumeration Type Documentation

5.80.2.1 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. 1012)

Enumerator:

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.

TOPIC_PRESENTATION_QOS Scope spans to all instances within the same **DDS::DataWriter** (p. 499) (or **DDS::DataReader** (p. 433)), but not across instances in different **DDS::DataWriter** (p. 499) (or **DDS::DataReader** (p. 433)).

GROUP_PRESENTATION_QOS [**Not supported (optional)**] Scope spans to all instances belonging to **DDS::DataWriter** (p. 499) (or **DDS::DataReader** (p. 433)) entities within the same **DDS::Publisher** (p. 1044) (or **DDS::Subscriber** (p. 1201)).

5.80.3 Function Documentation

5.80.3.1 static System::String ^ DDS::PresentationQosPolicy::get_presentation_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::PresentationQosPolicy** (p. 1012).

5.81 DEADLINE

Expresses the maximum duration (deadline) within which an instance is expected to be updated.

Classes

- struct **DDS::DeadlineQosPolicy**

  Expresses the maximum duration (deadline) within which an instance is expected to be updated.

Functions

- static System::String^ **DDS::DeadlineQosPolicy::get_deadline_qos_policy_name** ()

  *Stringified human-readable name for **DDS::DeadlineQosPolicy*** (p. 557).

5.81.1 Detailed Description

Expresses the maximum duration (deadline) within which an instance is expected to be updated.

5.81.2 Function Documentation

5.81.2.1 static System::String ^ DDS::DeadlineQosPolicy::get_deadline_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::DeadlineQosPolicy** (p. 557).

5.82 LATENCY_BUDGET

Provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.

Classes

- struct **DDS::LatencyBudgetQosPolicy**

  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.

Functions

- static System::String^ **DDS::LatencyBudgetQosPolicy::get_latencybudget_qos_policy_name** ()

  *Stringified human-readable name for **DDS::LatencyBudgetQosPolicy*** (p. 948).

5.82.1 Detailed Description

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.

5.82.2 Function Documentation

5.82.2.1 static System::String ^ DDS::LatencyBudgetQosPolicy::get_latencybudget_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::LatencyBudgetQosPolicy** (p. 948).

5.83 OWNERSHIP

Specifies whether it is allowed for multiple **DDS::DataWriter** (p. 499) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.

Classes

^ struct **DDS::OwnershipQosPolicy**

*Specifies whether it is allowed for multiple **DDS::DataWriter** (p. 499) (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.

Functions

^ static System::String^ **DDS::OwnershipQosPolicy::get_ownership_qos_policy_name** ()

*Stringified human-readable name for **DDS::OwnershipQosPolicy** (p. 993).*

5.83.1 Detailed Description

Specifies whether it is allowed for multiple **DDS::DataWriter** (p. 499) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.

5.83.2 Enumeration Type Documentation

5.83.2.1 enum DDS::OwnershipQosPolicyKind

Kinds of ownership.

QoS:

DDS::OwnershipQosPolicy (p. 993)

Enumerator:

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. 283) policy is not specified or supported.

EXCLUSIVE_OWNERSHIP_QOS Indicates each instance can only be owned by one **DDS::DataWriter** (p. 499), but the owner of an instance can change dynamically.

The selection of the owner is controlled by the setting of the **OWNERSHIP_STRENGTH** (p. 285) policy. The owner is always set to be the highest-strength **DDS::DataWriter** (p. 499) object among the ones currently active (as determined by the **LIVELINESS** (p. 286)).

5.83.3 Function Documentation

5.83.3.1 static System::String ^ DDS::OwnershipQosPolicy::get_ownership_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::OwnershipQosPolicy** (p. 993).

5.84 OWNERSHIP_STRENGTH

Specifies the value of the strength used to arbitrate among multiple **DDS::DataWriter** (p. 499) objects that attempt to modify the same instance of a data type (identified by **DDS::Topic** (p. 1258) + key).

Classes

- ^ struct **DDS::OwnershipStrengthQosPolicy**

  *Specifies the value of the strength used to arbitrate among multiple **DDS::DataWriter** (p. 499) objects that attempt to modify the same instance of a data type (identified by **DDS::Topic** (p. 1258) + key).*

Functions

- ^ static System::String^ **DDS::OwnershipStrengthQosPolicy::get_ownershipstrength_qos_policy_name** ()

  *Stringified human-readable name for **DDS::OwnershipStrengthQosPolicy** (p. 1000).*

5.84.1 Detailed Description

Specifies the value of the strength used to arbitrate among multiple **DDS::DataWriter** (p. 499) objects that attempt to modify the same instance of a data type (identified by **DDS::Topic** (p. 1258) + key).

5.84.2 Function Documentation

5.84.2.1 static System::String ^ DDS::OwnershipStrengthQosPolicy::get_ownershipstrength_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::OwnershipStrengthQosPolicy** (p. 1000).

5.85 LIVELINESS

Specifies and configures the mechanism that allows **DDS::DataReader** (p. 433) entities to detect when **DDS::DataWriter** (p. 499) entities become disconnected or "dead.".

Classes

- struct **DDS::LivelinessQosPolicy**

  *Specifies and configures the mechanism that allows **DDS::DataReader** (p. 433) entities to detect when **DDS::DataWriter** (p. 499) 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.

Functions

- static System::String^ **DDS::LivelinessQosPolicy::get_livelines_qos_policy_name** ()

  *Stringified human-readable name for **DDS::LivelinessQosPolicy** (p. 960).*

5.85.1 Detailed Description

Specifies and configures the mechanism that allows **DDS::DataReader** (p. 433) entities to detect when **DDS::DataWriter** (p. 499) entities become disconnected or "dead.".

5.85.2 Enumeration Type Documentation

5.85.2.1 enum DDS::LivelinessQosPolicyKind

Kinds of liveliness.

QoS:

DDS::LivelinessQosPolicy (p. 960)

Enumerator:

AUTOMATIC_LIVELINESS_QOS **[default]** The infrastructure will automatically signal liveliness for the **DDS::DataWriter** (p. 499) (s) at least as often as required by the `lease_duration`.

A **DDS::DataWriter** (p. 499) with this setting does not need to take any specific action in order to be considered 'alive.' The **DDS::DataWriter** (p. 499) 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.

MANUAL_BY_PARTICIPANT_LIVELINESS_QOS RTI Data Distribution Service will assume that as long as at least one **DDS::DataWriter** (p. 499) belonging to the **DDS::DomainParticipant** (p. 577) (or the **DDS::DomainParticipant** (p. 577) itself) has asserted its liveliness, then the other Entities belonging to that same **DDS::DomainParticipant** (p. 577) are also alive.

The user application takes responsibility to signal liveliness to RTI Data Distribution Service either by calling **DDS::DomainParticipant::assert_liveliness** (p. 637), or by calling **DDS::DataWriter::assert_liveliness** (p. 508), or **DDS::TypedDataWriter::write** (p. 1376) on any **DDS::DataWriter** (p. 499) belonging to the **DDS::DomainParticipant** (p. 577).

MANUAL_BY_TOPIC_LIVELINESS_QOS RTI Data Distribution Service will only assume liveliness of the **DDS::DataWriter** (p. 499) if the application has asserted liveliness of that **DDS::DataWriter** (p. 499) itself.

The user application takes responsibility to signal liveliness to RTI Data Distribution Service using the **DDS::DataWriter::assert_liveliness** (p. 508) method, or by writing some data.

5.85.3 Function Documentation

5.85.3.1 static System::String ^ DDS::LivelinessQosPolicy::get_livelines_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::LivelinessQosPolicy** (p. 960).

5.86 TIME_BASED_FILTER

Filter that allows a **DDS::DataReader** (p. 433) to specify that it is interested only in (potentially) a subset of the values of the data.

Classes

- struct **DDS::TimeBasedFilterQosPolicy**

  *Filter that allows a **DDS::DataReader** (p. 433) to specify that it is interested only in (potentially) a subset of the values of the data.*

Functions

- static System::String^ **DDS::TimeBasedFilterQosPolicy::get_timebasedfilter_qos_policy_name** ()

  *Stringified human-readable name for **DDS::TimeBasedFilterQosPolicy** (p. 1254).*

5.86.1 Detailed Description

Filter that allows a **DDS::DataReader** (p. 433) to specify that it is interested only in (potentially) a subset of the values of the data.

5.86.2 Function Documentation

5.86.2.1 static System::String ^ DDS::TimeBasedFilterQosPolicy::get_timebasedfilter_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::TimeBasedFilterQosPolicy** (p. 1254).

5.87 PARTITION

Set of strings that introduces a logical partition among the topics visible by a **DDS::Publisher** (p. 1044) and a **DDS::Subscriber** (p. 1201).

Classes

^ class **DDS::PartitionQosPolicy**

*Set of strings that introduces a logical partition among the topics visible by a **DDS::Publisher** (p. 1044) and a **DDS::Subscriber** (p. 1201).*

Functions

^ static System::String^ **DDS::PartitionQosPolicy::get_partition_qos_policy_name** ()

*Stringified human-readable name for **DDS::PartitionQosPolicy** (p. 1008).*

5.87.1 Detailed Description

Set of strings that introduces a logical partition among the topics visible by a **DDS::Publisher** (p. 1044) and a **DDS::Subscriber** (p. 1201).

5.87.2 Function Documentation

5.87.2.1 static System::String ^ DDS::PartitionQosPolicy::get_partition_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::PartitionQosPolicy** (p. 1008).

5.88 RELIABILITY

Indicates the level of reliability offered/requested by RTI Data Distribution Service.

Classes

- struct **DDS::ReliabilityQosPolicy**

  Indicates the level of reliability offered/requested by RTI Data Distribution Service.

Enumerations

- enum **DDS::ReliabilityQosPolicyKind** {
  DDS::BEST_EFFORT_RELIABILITY_QOS,
  DDS::RELIABLE_RELIABILITY_QOS }

  Kinds of reliability.

Functions

- static System::String^ **DDS::ReliabilityQosPolicy::get_reliability_qos_policy_name** ()

  *Stringified human-readable name for **DDS::ReliabilityQosPolicy*** *(p. 1094).*

5.88.1 Detailed Description

Indicates the level of reliability offered/requested by RTI Data Distribution Service.

5.88.2 Enumeration Type Documentation

5.88.2.1 enum DDS::ReliabilityQosPolicyKind

Kinds of reliability.

QoS:

DDS::ReliabilityQosPolicy (p. 1094)

Enumerator:

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. 433) and **DDS::Topic** (p. 1258)

RELIABLE_RELIABILITY_QOS Specifies RTI Data Distribution Service 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. 499)) RTI Data Distribution Service guarantees that all samples in the **DDS::DataWriter** (p. 499) history will eventually be delivered to all the **DDS::DataReader** (p. 433) objects (subject to timeouts that indicate loss of communication with a particular **DDS::Subscriber** (p. 1201)).

Outside steady state the **HISTORY** (p. 294) and **RESOURCE_LIMITS** (p. 298) policies will determine how samples become part of the history and whether samples can be discarded from it.

[**default**] for **DDS::DataWriter** (p. 499)

5.88.3 Function Documentation

5.88.3.1 static System::String ^ DDS::ReliabilityQosPolicy::get_reliability_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::ReliabilityQosPolicy** (p. 1094).

5.89 DESTINATION_ORDER

Controls the criteria used to determine the logical order among changes made by **DDS::Publisher** (p. 1044) entities to the same instance of data (i.e., matching **DDS::Topic** (p. 1258) and key).

Classes

- struct **DDS::DestinationOrderQosPolicy**

  *Controls how the middleware will deal with data sent by multiple **DDS::DataWriter** (p. 499) entities for the same instance of data (i.e., same **DDS::Topic** (p. 1258) and key).*

Enumerations

- enum **DDS::DestinationOrderQosPolicyKind** {

  DDS::BY_RECEPTION_TIMESTAMP_-DESTINATIONORDER_QOS,

  DDS::BY_SOURCE_TIMESTAMP_DESTINATIONORDER_-QOS }

  Kinds of destination order.

Functions

- static System::String^ **DDS::DestinationOrderQosPolicy::get_-destinationorder_qos_policy_name** ()

  *Stringified human-readable name for **DDS::DestinationOrderQosPolicy** (p. 560).*

5.89.1 Detailed Description

Controls the criteria used to determine the logical order among changes made by **DDS::Publisher** (p. 1044) entities to the same instance of data (i.e., matching **DDS::Topic** (p. 1258) and key).

5.89.2 Enumeration Type Documentation

5.89.2.1 enum DDS::DestinationOrderQosPolicyKind

Kinds of destination order.

QoS:

DDS::DestinationOrderQosPolicy (p. 560)

Enumerator:

BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS
[**default**] Indicates that data is ordered based on the reception time at each **DDS::Subscriber** (p. 1201).

Since each subscriber may receive the data at different times there is no guaranteed 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.

BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS
Indicates that data is ordered based on a time-stamp placed at the source (by RTI Data Distribution Service or by the application).

In any case this guarantees a consistent final value for the data in all subscribers.

See also:

Special Instructions if Using Timestamp APIs and BY_SOURCE_TIMESTAMP Destination Ordering: (p. 1057)

5.89.3 Function Documentation

5.89.3.1 static System::String ^ DDS::DestinationOrderQosPolicy::get_destinationorder_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::DestinationOrderQosPolicy** (p. 560).

5.90 HISTORY

Specifies the behavior of RTI Data Distribution Service in the case where the value of an instance changes (one or more times) before it can be successfully communicated to one or more existing subscribers.

Classes

- struct **DDS::HistoryQosPolicy**

  Specifies the behavior of RTI Data Distribution Service 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.

- enum **DDS::RefilterQosPolicyKind** {
  DDS::NONE_REFILTER_QOS,
  DDS::ALL_REFILTER_QOS,
  DDS::ON_DEMAND_REFILTER_QOS }

  <<**eXtension**>> (p. 174) *Kinds of Refiltering*

Functions

- static System::String^ **DDS::HistoryQosPolicy::get_history_qos_policy_name** ()

  *Stringified human-readable name for **DDS::HistoryQosPolicy*** (p. 898).

5.90.1 Detailed Description

Specifies the behavior of RTI Data Distribution Service 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.

5.90.2 Enumeration Type Documentation

5.90.2.1 enum DDS::HistoryQosPolicyKind

Kinds of history.

QoS:

DDS::HistoryQosPolicy (p. 898)

Enumerator:

KEEP_LAST_HISTORY_QOS [**default**] Keep the last `depth` samples.

On the publishing side, RTI Data Distribution Service will only attempt to keep the most recent `depth` samples of each instance of data (identified by its key) managed by the **DDS::DataWriter** (p. 499).

On the subscribing side, the **DDS::DataReader** (p. 433) will only attempt to keep the most recent `depth` samples received for each instance (identified by its key) until the application takes them via the **DDS::DataReader** (p. 433) 's **take()** operation.

KEEP_ALL_HISTORY_QOS Keep *all* the samples.

On the publishing side, RTI Data Distribution Service 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. 499) until they can be delivered to all subscribers.

On the subscribing side, RTI Data Distribution Service will attempt to keep all samples of each instance of data (identified by its key) managed by the **DDS::DataReader** (p. 433). These samples are kept until the application takes them from RTI Data Distribution Service via the **take()** operation.

5.90.2.2 enum DDS::RefilterQosPolicyKind

<<***eXtension***>> *(p. 174)* Kinds of Refiltering

QoS:

DDS::HistoryQosPolicy (p. 898)

Enumerator:

NONE_REFILTER_QOS [**default**] Do not filter existing samples for a new reader

On the publishing side, when a new reader matches a writer, the writer can be configured to filter previously written samples stored in the writer queue for the new reader. This option configures the writer to not filter any existing samples for the reader and the reader will do the filtering.

ALL_REFILTER_QOS Filter all existing samples for a new reader.

On the publishing side, when a new reader matches a writer, the writer can be configured to filter previously written samples stored in the writer queue. This option configures the writer to filter all existing samples for the reader when a new reader is matched to the writer.

ON_DEMAND_REFILTER_QOS Filter existing samples only when they are requested by the reader.

On the publishing side, when a new reader matches a writer, the writer can be configured to filter previously written samples stored in the writer queue. This option configures the writer to filter only existing samples that are requested by the reader.

5.90.3 Function Documentation

5.90.3.1 static System::String ^ DDS::HistoryQosPolicy::get_history_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::HistoryQosPolicy** (p. 898).

5.91 DURABILITY_SERVICE

Various settings to configure the external *RTI Persistence Service* used by RTI Data Distribution Service for DataWriters with a **DDS::DurabilityQosPolicy** (p. 709) setting of DDS::DurabilityQosPolicyKind::PERSISTENT_DURABILITY_QOS or DDS::DurabilityQosPolicyKind::TRANSIENT_DURABILITY_QOS.

Classes

^ struct **DDS::DurabilityServiceQosPolicy**

Various settings to configure the external RTI Persistence Service *used by RTI Data Distribution Service for DataWriters with a* ***DDS::DurabilityQosPolicy*** *(p. 709) setting of DDS::DurabilityQosPolicyKind::PERSISTENT_DURABILITY_QOS or DDS::DurabilityQosPolicyKind::TRANSIENT_DURABILITY_QOS.*

Functions

^ static System::String^ **DDS::DurabilityServiceQosPolicy::get_durabilityservice_qos_policy_name** ()

Stringified human-readable name for ***DDS::DurabilityServiceQosPolicy*** *(p. 714).*

5.91.1 Detailed Description

Various settings to configure the external *RTI Persistence Service* used by RTI Data Distribution Service for DataWriters with a **DDS::DurabilityQosPolicy** (p. 709) setting of DDS::DurabilityQosPolicyKind::PERSISTENT_DURABILITY_QOS or DDS::DurabilityQosPolicyKind::TRANSIENT_DURABILITY_QOS.

5.91.2 Function Documentation

5.91.2.1 static System::String ^ DDS::DurabilityServiceQosPolicy::get_durabilityservice_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::DurabilityServiceQosPolicy** (p. 714).

5.92 RESOURCE_LIMITS

Controls the amount of physical memory allocated for DDS entities, if dynamic allocations are allowed, and how they occur. Also controls memory usage among different instance values for keyed topics.

Classes

- struct **DDS::ResourceLimitsQosPolicy**

  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.

Functions

- static System::String^ **DDS::ResourceLimitsQosPolicy::get_resourcelimits_qos_policy_name** ()

  *Stringified human-readable name for **DDS::ResourceLimitsQosPolicy*** *(p. 1109).*

Properties

- static System::Int32 **DDS::ResourceLimitsQosPolicy::LENGTH_UNLIMITED** `[get]`

  A special value indicating an unlimited quantity.

5.92.1 Detailed Description

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.

5.92.2 Function Documentation

5.92.2.1 static System::String ^ DDS::ResourceLimitsQosPolicy::get_resourcelimits_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::ResourceLimitsQosPolicy** (p. 1109).

5.92.3 Properties

5.92.3.1 System:: Int32 DDS::ResourceLimitsQosPolicy::LENGTH_UNLIMITED [static, get, inherited]

A special value indicating an unlimited quantity.

Examples:

HelloWorld_subscriber.cpp.

5.93 TRANSPORT_PRIORITY

This QoS policy allows the application to take advantage of transports that are capable of sending messages with different priorities.

Classes

- struct **DDS::TransportPriorityQosPolicy**

  This QoS policy allows the application to take advantage of transports that are capable of sending messages with different priorities.

Functions

- static System::String^ **DDS::TransportPriorityQosPolicy::get_transportpriority_qos_policy_name** ()

  *Stringified human-readable name for **DDS::TransportPriorityQosPolicy*** (p. 1292).

5.93.1 Detailed Description

This QoS policy allows the application to take advantage of transports that are capable of sending messages with different priorities.

5.93.2 Function Documentation

5.93.2.1 static System::String ^ DDS::TransportPriorityQosPolicy::get_transportpriority_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::TransportPriorityQosPolicy** (p. 1292).

5.94 LIFESPAN

Specifies how long the data written by the **DDS::DataWriter** (p. 499) is considered valid.

Classes

- struct **DDS::LifespanQosPolicy**

  *Specifies how long the data written by the **DDS::DataWriter** (p. 499) is considered valid.*

Functions

- static System::String^ **DDS::LifespanQosPolicy::get_lifespan_qos_policy_name** ()

  *Stringified human-readable name for **DDS::LifespanQosPolicy** (p. 950).*

5.94.1 Detailed Description

Specifies how long the data written by the **DDS::DataWriter** (p. 499) is considered valid.

5.94.2 Function Documentation

5.94.2.1 static System::String ^ DDS::LifespanQosPolicy::get_lifespan_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::LifespanQosPolicy** (p. 950).

5.95 WRITER_DATA_LIFECYCLE

Controls how a **DataWriter** (p. 499) handles the lifecycle of the instances (keys) that it is registered to manage.

Classes

- struct **DDS::WriterDataLifecycleQosPolicy**

  *Controls how a **DDS::DataWriter** (p. 499) handles the lifecycle of the instances (keys) that it is registered to manage.*

Functions

- static System::String^ **DDS::WriterDataLifecycleQosPolicy::get_writerdatalifecycle_qos_policy_name** ()

  *Stringified human-readable name for **DDS::WriterDataLifecycleQosPolicy** (p. 1431).*

5.95.1 Detailed Description

Controls how a **DataWriter** (p. 499) handles the lifecycle of the instances (keys) that it is registered to manage.

5.95.2 Function Documentation

5.95.2.1 static System::String ^ DDS::WriterDataLifecycleQosPolicy::get_writerdatalifecycle_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::WriterDataLifecycleQosPolicy** (p. 1431).

5.96 READER_DATA_LIFECYCLE

Controls how a **DataReader** (p. 433) manages the lifecycle of the data that it has received.

Classes

- ^ struct **DDS::ReaderDataLifecycleQosPolicy**

  *Controls how a **DataReader** (p. 433) manages the lifecycle of the data that it has received.*

Functions

- ^ static System::String^ **DDS::ReaderDataLifecycleQosPolicy::get_readerdatalifecycle_qos_policy_name** ()

  *Stringified human-readable name for **DDS::ReaderDataLifecycleQosPolicy** (p. 1087).*

5.96.1 Detailed Description

Controls how a **DataReader** (p. 433) manages the lifecycle of the data that it has received.

5.96.2 Function Documentation

5.96.2.1 static System::String ^ DDS::ReaderDataLifecycleQosPolicy::get_readerdatalifecycle_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::ReaderDataLifecycleQosPolicy** (p. 1087).

5.97 ENTITY_FACTORY

A QoS policy for all **DDS::Entity** (p. 845) types that can act as factories for one or more other **DDS::Entity** (p. 845) types.

Classes

- struct **DDS::EntityFactoryQosPolicy**

  *A QoS policy for all **DDS::Entity** (p. 845) types that can act as factories for one or more other **DDS::Entity** (p. 845) types.*

Functions

- static System::String^ **DDS::EntityFactoryQosPolicy::get_entityfactory_qos_policy_name** ()

  *Stringified human-readable name for **DDS::EntityFactoryQosPolicy** (p. 851).*

5.97.1 Detailed Description

A QoS policy for all **DDS::Entity** (p. 845) types that can act as factories for one or more other **DDS::Entity** (p. 845) types.

5.97.2 Function Documentation

5.97.2.1 static System::String ^ DDS::EntityFactoryQosPolicy::get_entityfactory_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::EntityFactoryQosPolicy** (p. 851).

5.98 Extended Qos Support

<<***eXtension***>> *(*p. *174)* Types and defines used in extended QoS policies.

Modules

- **Thread Settings**

  The properties of a thread of execution.

Classes

- struct **DDS::RtpsReliableReaderProtocol_t**

  Qos related to reliable reader protocol defined in RTPS.

- struct **DDS::RtpsReliableWriterProtocol_t**

  QoS related to the reliable writer protocol defined in RTPS.

5.98.1 Detailed Description

<<***eXtension***>> *(*p. *174)* Types and defines used in extended QoS policies.

5.99 Unicast Settings

Unicast communication settings.

Classes

- class **DDS::TransportUnicastSettings_t**

  Type representing a list of unicast locators.

- class **DDS::TransportUnicastSettingsSeq**

  Declares IDL `sequence<` ***DDS::TransportUnicastSettings_t*** *(p. 1298)* `>`.

5.99.1 Detailed Description

Unicast communication settings.

5.100 Multicast Settings

Multicast communication settings.

Classes

- ^ class **DDS::TransportMulticastSettings_t**

  Type representing a list of multicast locators.

- ^ class **DDS::TransportMulticastSettingsSeq**

  Declares IDL `sequence<` ***DDS::TransportMulticastSettings_t*** *(*p. 1289*)* `>`.

5.100.1 Detailed Description

Multicast communication settings.

5.101 TRANSPORT_SELECTION

<<*eXtension*>> *(p. 174)* Specifies the physical transports a **DDS::DataWriter** (p. 499) or **DDS::DataReader** (p. 433) may use to send or receive data.

Classes

- class **DDS::TransportSelectionQosPolicy**

  *Specifies the physical transports a **DDS::DataWriter** (p. 499) or **DDS::DataReader** (p. 433) may use to send or receive data.*

Functions

- static System::String^ **DDS::TransportSelectionQosPolicy::get_transportselection_qos_policy_name** ()

  *Stringified human-readable name for **DDS::TransportSelectionQosPolicy** (p. 1294).*

5.101.1 Detailed Description

<<*eXtension*>> *(p. 174)* Specifies the physical transports a **DDS::DataWriter** (p. 499) or **DDS::DataReader** (p. 433) may use to send or receive data.

5.101.2 Function Documentation

5.101.2.1 static System::String ^ DDS::TransportSelectionQosPolicy::get_transportselection_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::TransportSelectionQosPolicy** (p. 1294).

5.102 TRANSPORT_UNICAST

<<*eXtension*>> *(p. 174)* Specifies a subset of transports and a port number that can be used by an **Entity** (p. 845) to receive data.

Modules

- **Unicast Settings**

  Unicast communication settings.

Classes

- class **DDS::TransportUnicastQosPolicy**

  *Specifies a subset of transports and a port number that can be used by an **Entity** (p. 845) to receive data.*

Functions

- static System::String^ **DDS::TransportUnicastQosPolicy::get_transportunicast_qos_policy_name** ()

  *Stringified human-readable name for **DDS::TransportUnicastQosPolicy** (p. 1296).*

5.102.1 Detailed Description

<<*eXtension*>> *(p. 174)* Specifies a subset of transports and a port number that can be used by an **Entity** (p. 845) to receive data.

5.102.2 Function Documentation

5.102.2.1 static System::String ^ DDS::TransportUnicastQosPolicy::get_transportunicast_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::TransportUnicastQosPolicy** (p. 1296).

5.103 TRANSPORT_MULTICAST

<<*eXtension*>> *(p. 174)* Specifies the multicast address on which a **DDS::DataReader** (p. 433) 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. 577) level) transports with which to receive the multicast data.

Modules

- **Multicast Settings**

  Multicast communication settings.

Classes

- class **DDS::TransportMulticastQosPolicy**

  *Specifies the multicast address on which a **DDS::DataReader** (p. 433) 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. 577) level) transports with which to receive the multicast data.*

Functions

- static System::String^ **DDS::TransportMulticastQosPolicy::get_transportmulticast_qos_policy_name** ()

  *Stringified human-readable name for **DDS::TransportMulticastQosPolicy** (p. 1287).*

5.103.1 Detailed Description

<<*eXtension*>> *(p. 174)* Specifies the multicast address on which a **DDS::DataReader** (p. 433) 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. 577) level) transports with which to receive the multicast data.

5.103.2 Function Documentation

5.103.2.1 static System::String ^ DDS::TransportMulticastQosPolicy::get_transportmulticast_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::TransportMulticastQosPolicy** (p. 1287).

5.104 NDDS_DISCOVERY_PEERS

Environment variable or a file that specifies the default values of **DDS::DiscoveryQosPolicy::initial_peers** (p. 573) and **DDS::DiscoveryQosPolicy::multicast_receive_addresses** (p. 573) contained in the **DDS::DomainParticipantQos::discovery** (p. 686) qos policy.

The default value of the **DDS::DomainParticipantQos** (p. 683) is obtained by calling **DDS::DomainParticipantFactory::get_default_participant_qos()** (p. 656).

NDDS_DISCOVERY_PEERS specifies the default value of the **DDS::DiscoveryQosPolicy::initial_peers** (p. 573) and **DDS::DiscoveryQosPolicy::multicast_receive_addresses** (p. 573) fields, when the default participant QoS policies have not been explictly set by the user (i.e., **DDS::DomainParticipantFactory::set_default_participant_qos()** (p. 654) has never been called or was called using **DDS::DomainParticipantFactory::PARTICIPANT_QOS_DEFAULT** (p. 35)).

If NDDS_DISCOVERY_PEERS does *not* contain a multicast address, then the string sequence **DDS::DiscoveryQosPolicy::multicast_receive_addresses** (p. 573) 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. 573), starting at element 0. They will be stored in the order in which they appear in NDDS_DISCOVERY_PEERS.

Note: IPv4 multicast addresses must have a prefix. Therefore, when using the UDPv6 transport: if there are any IPv4 multicast addresses in the peers list, make sure they have "udpv4://" in front of them (such as udpv4://239.255.0.1).

Note: Currently, RTI Data Distribution Service will only listen for discovery traffic on the first multicast address (element 0) in **DDS::DiscoveryQosPolicy::multicast_receive_addresses** (p. 573).

NDDS_DISCOVERY_PEERS provides a mechanism to dynamically switch the discovery configuration of an RTI Data Distribution Service 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. 313)) or in a file. These formats are described below.

5.104.1 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. 573) field and the **DDS::DomainParticipant::add_peer()** (p. 644) operation.

The anatomy of a `peer` descriptor is illustrated below using a special "StarFabric" transport example.

A peer descriptor consists of:

optional **Maximum Participant ID**. Specifies the maximum participant ID that is contacted by the RTI Data Distribution Service discovery mechanism at the given locator. If omitted, a default value of 4 is implied.

- ^ **Locator**. See **Locator Format** (p. 313).

These are separated by the '@' character. The separator may be omitted if a participant ID limit is not explictly specified.

Note that the "participant ID limit" only applies to unicast locators, and is ignored for multicast locators (and therefore should be omitted for multicast peer descriptors).

5.104.1.1 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. 114)) that may be used to parse the `address` portion of the locator. Note that a transport class name is an implicit alias that used to refer to all the transport plugin instances of that class.

optional **Address**. See **Address Format** (p. 314).

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. 845)) may be used to parse the address string.

5.104.1.2 Address Format

An **address** string specifies a transport independent network address that qualifies an **transport dependent** address string. Addresses are used to form locators. Addresses are also used in **DDS::DiscoveryQosPolicy::multicast_receive_addresses** (p. 573), and **DDS::TransportMulticastSettings_t::receive_address** (p. 1290) 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. 117)).

optional **Transport Address**. A string that is passed to the transport for processing. The transport maps this string into DDS::Transport_Property_t::address_bit_count 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.

The bits resulting from the transport address string are prepended with the network address. The least significant DDS::Transport_Property_t::address_bit_count bits of the network address are ignored (**Transport Network Address** (p. 117)).

If the separator is omitted and the string is not a valid IPv4 or IPv6 address, it is treated as a transport address with an implicit network address (of the transport plugin).

5.104.2 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 FAA0::0#localhost (could be a UDPv4 transport plugin registered at network address of FAA0::0) (IPv6)

- FAA0::0#localhost

Default maximum participant ID on host himalaya accessed using the "udpv4" transport plugin(s) (IPv4)

- udpv4://himalaya

Default maximum participant ID on localhost using the "udpv4" transport plugin(s) registered at network address FAA0::0

- udpv4://FAA0::0#localhost

Default maximum participant ID on all unicast addresses accessed via the "udpv4" (UDPv4) transport plugin(s)

- udpv4://

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)

- shmem://

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

5.104.3 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 - Default 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 RTI Data Distribution Service discovery multicast address

;; Unicast
localhost,192.168.1.1       ; A comma can be used a separator

FAA0::1  FAA0::0#localhost  ; Whitespace can be used as a separator

1@himalaya                  ; Maximum participant ID of 1 on 'himalaya'
1@gangotri


;; UDPv4
udpv4://himalaya            ; 'himalaya' via 'udpv4' transport plugin(s)
udpv4://FAA0::0#localhost   ; 'localhost' via 'updv4' transport
                            ;  plugin registered at network address FAA0::0

;; Shared Memory
shmem://                    ; All 'shmem' transport plugin(s)
builtin.shmem://            ; The builtin 'shmem' transport plugin
shmem://FCC0::0             ; Shared memory transport plugin registered
                            ;                at network address FCC0::0

;; StarFabric
0/0/R                       ; StarFabric node 0/0/R
starfabric://0/0/R          ; 0/0/R accessed via 'starfabric'
                            ;             transport plugin(s)
starfabric://FBB0::0#0/0/R  ; StarFabric transport plugin registered
                            ;             at network address FBB0::0
starfabric://               ; All 'starfabric' transport plugin(s)
```

5.104.4 NDDS_DISCOVERY_PEERS Precedence

If the current working directory from which the RTI Data Distribution Service 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.

5.104.5 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 RTI Data Distribution Service applications anywhere on the local network to
;; discover each other over UDPv4.

builtin.udpv4://239.255.0.1  ; RTI Data Distribution Service's default discovery multicast address


;; Unicast - UDPv4 (on all platforms)
;;
;; This allows two RTI Data Distribution Service 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 RTI Data Distribution Service 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://
```

5.104.6 Builtin Transport Class Names

The class names for the builtin transport plugins are:

- `shmem` - **DDS::ShmemTransport** (p. 1177)
- `udpv4` - **DDS::UDPv4Transport** (p. 1388)
- `udpv6` - **DDS::UDPv6Transport** (p. 1391)

These may be used as the transport names in the **Locator Format** (p. 313).

5.104.7 NDDS_DISCOVERY_PEERS and Local Host Communication

Suppose you want to communicate with other RTI Data Distribution Service 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. 1285) of the **DDS::DomainParticipant** (p. 577). 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 document.)

See also:

DDS::DiscoveryQosPolicy::multicast_receive_addresses (p. 573)
DDS::DiscoveryQosPolicy::initial_peers (p. 573)
DDS::DomainParticipant::add_peer() (p. 644)
DDS::DomainParticipantFactory::PARTICIPANT_QOS_DEFAULT (p. 35)
DDS::DomainParticipantFactory::get_default_participant_qos() (p. 656)
Transport Aliases (p. 114)
Transport Network Address (p. 117)

5.105 DISCOVERY

<<***eXtension***>> *(p. 174)* 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. 573) and* ***DDS::DiscoveryQosPolicy::multicast_receive_addresses*** *(p. 573) contained in the* ***DDS::DomainParticipantQos::discovery*** *(p. 686) qos policy.*

Classes

- class **DDS::DiscoveryQosPolicy**

  Configures the mechanism used by the middleware to automatically discover and connect with new remote applications.

Functions

- static System::String^ **DDS::DiscoveryQosPolicy::get_discovery_qos_policy_name** ()

  Stringified human-readable name for ***DDS::DiscoveryQosPolicy*** *(p. 571).*

5.105.1 Detailed Description

<<***eXtension***>> *(p. 174)* Specifies the attributes required to discover participants in the domain.

5.105.2 Function Documentation

5.105.2.1 static System::String ^ DDS::DiscoveryQosPolicy::get_discovery_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::DiscoveryQosPolicy** (p. 571).

5.106 TRANSPORT_BUILTIN

<<*eXtension*>> *(*p. *174)* Specifies which built-in transports are used.

Classes

- class **DDS::TransportBuiltinKindAlias**

  Bits in DDS::TransportBuiltinKindMask .

- struct **DDS::TransportBuiltinQosPolicy**

  Specifies which built-in transports are used.

Enumerations

- enum **DDS::TransportBuiltinKind** {
  DDS::TRANSPORTBUILTIN_UDPv4,
  DDS::TRANSPORTBUILTIN_SHMEM ,
  DDS::TRANSPORTBUILTIN_UDPv6 }

  Built-in transport kind.

- enum **DDS::TransportBuiltinKindMask** {
  DDS::TRANSPORTBUILTIN_MASK_NONE,
  DDS::TRANSPORTBUILTIN_MASK_DEFAULT,
  DDS::TRANSPORTBUILTIN_MASK_ALL }

  A mask of DDS::TransportBuiltinKind bits.

Functions

- static System::String^ **DDS::TransportBuiltinQosPolicy::get_transportbuiltin_qos_policy_name** ()

  Stringified human-readable name for ***DDS::TransportBuiltinQosPolicy*** *(*p. *1285).*

Variables

- static System::String^ **DDS::TransportBuiltinKindAlias::TRANSPORTBUILTIN_SHMEM_ALIAS**

Alias name for the shared memory built-in transport.

^ static System::String^ **DDS::TransportBuiltinKindAlias::TRANSPORTBUILTIN_UDPv4_ALIAS**

Alias name for the UDPv4 built-in transport.

^ static System::String^ **DDS::TransportBuiltinKindAlias::TRANSPORTBUILTIN_UDPv6_ALIAS**

Alias name for the UDPv6 built-in transport.

5.106.1 Detailed Description

<<***eXtension***>> *(*p. *174**)* Specifies which built-in transports are used.

See also:

Changing the automatically registered built-in transports (p. 165)

5.106.2 Enumeration Type Documentation

5.106.2.1 enum DDS::TransportBuiltinKind

Built-in transport kind.

See also:

DDS::TransportBuiltinKindMask

Enumerator:

TRANSPORTBUILTIN_UDPv4 Built-in UDPv4 transport, **DDS::UDPv4Transport** (p. 1388).

TRANSPORTBUILTIN_SHMEM Built-in shared memory transport, **DDS::ShmemTransport** (p. 1177).

TRANSPORTBUILTIN_UDPv6 Built-in UDPv6 transport, **DDS::UDPv6Transport** (p. 1391).

5.106.2.2 enum DDS::TransportBuiltinKindMask

A mask of DDS::TransportBuiltinKind bits.

QoS:

DDS::TransportBuiltinQosPolicy (p. 1285)

Enumerator:

TRANSPORTBUILTIN_MASK_NONE None of the built-in transports will be registered automatically when the **DDS::DomainParticipant** (p. 577) is enabled. The user must explictly register transports using DDS::Transport_Support::register_transport.

See also:

DDS::TransportBuiltinKindMask

TRANSPORTBUILTIN_MASK_DEFAULT The default value of **DDS::TransportBuiltinQosPolicy::mask** (p. 1286).

The set of builtin transport plugins that will be automatically registered with the participant by default. The user can register additional transports using DDS::Transport_Support::register_transport.

See also:

DDS::TransportBuiltinKindMask

TRANSPORTBUILTIN_MASK_ALL All the available built-in transports are registered automatically when the **DDS::DomainParticipant** (p. 577) is enabled.

See also:

DDS::TransportBuiltinKindMask

5.106.3 Function Documentation

5.106.3.1 static System::String ^ DDS::TransportBuiltinQosPolicy::get_transportbuiltin_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::TransportBuiltinQosPolicy** (p. 1285).

5.106.4 Variable Documentation

5.106.4.1 System::String ^ DDS::TransportBuiltinKindAlias::TRANSPORTBUILTIN_SHMEM_ALIAS [static, inherited]

Alias name for the shared memory built-in transport.

5.106.4.2 System::String ^ DDS::TransportBuiltinKindAlias::TRANSPORTBUILTIN_UDPv4_ALIAS [static, inherited]

Alias name for the UDPv4 built-in transport.

5.106.4.3 System::String ^ DDS::TransportBuiltinKindAlias::TRANSPORTBUILTIN_UDPv6_ALIAS [static, inherited]

Alias name for the UDPv6 built-in transport.

5.107 WIRE_PROTOCOL

*<<**eXtension**>> (p. 174)* Specifies the wire protocol related attributes for the **DDS::DomainParticipant** (p. 577).

Classes

- struct **DDS::RtpsWellKnownPorts_t**

  RTPS well-known port mapping configuration.

- struct **DDS::WireProtocolQosPolicy**

  *Specifies the wire-protocol-related attributes for the **DDS::DomainParticipant** (p. 577).*

Enumerations

- enum **DDS::RtpsReservedPortKind** {

  DDS::DDS_RTPS_RESERVED_PORT_BUILTIN_UNICAST = 0x0001 << 0,

  DDS::DDS_RTPS_RESERVED_PORT_BUILTIN_MULTICAST = 0x0001 << 1,

  DDS::DDS_RTPS_RESERVED_PORT_USER_UNICAST = 0x0001 << 2,

  DDS::DDS_RTPS_RESERVED_PORT_USER_MULTICAST = 0x0001 << 3,

  DDS::RTPS_RESERVED_PORT_MASK_DEFAULT,

  DDS::RTPS_RESERVED_PORT_MASK_NONE,

  DDS::RTPS_RESERVED_PORT_MASK_ALL }

  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 }

  Kind of auto mechanism used to calculate the GUID prefix.

Functions

- static System::String^ **DDS::WireProtocolQosPolicy::get_wireprotocol_qos_policy_name** ()

  *Stringified human-readable name for **DDS::WireProtocolQosPolicy*** *(p. 1423).*

Properties

- static **RtpsWellKnownPorts_t DDS::RtpsWellKnownPorts_t::RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS** `[get]`

  Assign to use well-known port mappings which are compatible with previous versions of the RTI Data Distribution Service middleware.

- static **RtpsWellKnownPorts_t DDS::RtpsWellKnownPorts_t::INTEROPERABLE_RTPS_WELL_KNOWN_PORTS** `[get]`

  Assign to use well-known port mappings which are compliant with OMG's DDS Interoperability Wire Protocol.

- static System::UInt32 **DDS::WireProtocolQosPolicy::RTPS_AUTO_ID** `[get]`

  Indicates that RTI Data Distribution Service should choose an appropriate host, app, instance or object ID automatically.

5.107.1 Detailed Description

<<***eXtension***>> *(p. 174)* Specifies the wire protocol related attributes for the **DDS::DomainParticipant** (p. 577).

5.107.2 Enumeration Type Documentation

5.107.2.1 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. 1430)

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.

RTPS_RESERVED_PORT_MASK_DEFAULT The default value of **DDS::WireProtocolQosPolicy::rtps_reserved_port_mask** (p. 1430).

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. 1142) 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

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

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. 1142) will be detected at this time, and the enable operation will fail.

Note that this will also reserve the **usertraffic** multicast port which is not actually used unless there are DataReaders that enable multicast

but fail to specify a port. To avoid unnecesary resource usage for these ports, use RTPS_RESERVED_PORT_MASK_DEFAULT.

Automatic participant ID selection will be based on finding a participant index with both the discovery (metatraffic) unicast port and usertraffic unicast port available.

See also:

DDS::RtpsReservedPortKindMask

5.107.2.2 enum DDS::WireProtocolQosPolicyAutoKind

Kind of auto mechanism used to calculate the GUID prefix.

See also:

DDS::WireProtocolQosPolicy::rtps_auto_id_kind (p. 1430)

Enumerator:

RTPS_AUTO_ID_FROM_IP Select the **IPv4** based algorithm.

RTPS_AUTO_ID_FROM_MAC Select the **MAC** based algorithm.

Note to Solaris Users: To use DDS_RTPS_AUTO_ID_FROM_MAC, you must run the RTI Data Distribution Service application while logged in as root.

5.107.3 Function Documentation

5.107.3.1 static System::String ^ DDS::WireProtocolQosPolicy::get_wireprotocol_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::WireProtocolQosPolicy** (p. 1423).

5.107.4 Properties

5.107.4.1 RtpsWellKnownPorts_t DDS::RtpsWellKnownPorts_t::RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS [static, get, inherited]

Assign to use well-known port mappings which are compatible with previous versions of the RTI Data Distribution Service middleware.

Assign **DDS::WireProtocolQosPolicy::rtps_well_known_ports** (p. 1430) to this value to remain compatible with previous versions of the RTI Data Distribution Service middleware that used fixed port mappings.

The following are the `rtps_well_known_ports` values for **DDS::RtpsWellKnownPorts_t::RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS** (p. 328):

`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::RtpsWellKnownPorts_t::INTEROPERABLE_RTPS_WELL_KNOWN_PORTS** (p. 329).

See also:

DDS::WireProtocolQosPolicy::rtps_well_known_ports (p. 1430)
DDS::RtpsWellKnownPorts_t::INTEROPERABLE_RTPS_WELL_KNOWN_PORTS (p. 329)

5.107.4.2 RtpsWellKnownPorts_t DDS::RtpsWellKnownPorts_t::INTEROPERABLE_RTPS_WELL_KNOWN_PORTS [static, get, inherited]

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. 1430) 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::RtpsWellKnownPorts_t::INTEROPERABLE_RTPS_WELL_KNOWN_PORTS** (p. 329):

`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 Data Distribution Service middleware that used fixed port mappings. For backwards compability, please use **DDS::RtpsWellKnownPorts_t::RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS** (p. 328).

See also:

DDS::WireProtocolQosPolicy::rtps_well_known_ports (p. 1430)
DDS::RtpsWellKnownPorts_t::RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS (p. 328)

5.107.4.3 static System::UInt32 DDS::WireProtocolQosPolicy::RTPS_AUTO_ID [static, get, inherited]

Indicates that RTI Data Distribution Service should choose an appropriate host, app, instance or object ID automatically.

If this special value is assigned to **DDS::WireProtocolQosPolicy::rtps_host_id** (p. 1428), **DDS::WireProtocolQosPolicy::rtps_app_id** (p. 1429), **DDS::WireProtocolQosPolicy::rtps_instance_id** (p. 1429), **DDS::DataWriterProtocolQosPolicy::rtps_object_id** (p. 531) or **DDS::DataReaderProtocolQosPolicy::rtps_object_id** (p. 467) RTI Data Distribution Service will assign the ID automatically.

The actual ID value is chosen when the QoS is set: the QoS returned from **DDS::DomainParticipant::get_qos** (p. 643), **DDS::DataWriter::get_qos** (p. 514) or **DDS::DataReader::get_qos** (p. 449) will never have this value.

QoS:

DDS::WireProtocolQosPolicy::rtps_host_id (p. 1428)
DDS::WireProtocolQosPolicy::rtps_app_id (p. 1429)
DDS::WireProtocolQosPolicy::rtps_instance_id (p. 1429)

5.108 DATA_READER_RESOURCE_LIMITS

<<***eXtension***>> *(*p. *174)* Various settings that configure how DataReaders allocate and use physical memory for internal resources.

Classes

^ struct **DDS::DataReaderResourceLimitsQosPolicy**

Various settings that configure how a ***DDS::DataReader*** *(*p. *433) allocates and uses physical memory for internal resources.*

Functions

^ static System::String^ **DDS::DataReaderResourceLimitsQosPolicy::get_datareaderresourcelimits_qos_policy_name** ()

Stringified human-readable name for ***DDS::DataReaderResourceLimitsQosPolicy*** *(*p. *486).*

Properties

^ static System::Int32 **DDS::DataReaderResourceLimitsQosPolicy::AUTO_MAX_TOTAL_INSTANCES** `[get]`

<<**eXtension**>> (p. 174) *This value is used to make* ***DDS::DataReaderResourceLimitsQosPolicy::max_total_instances*** *(*p. *494) equal to* ***DDS::ResourceLimitsQosPolicy::max_instances*** *(*p. *1112).*

5.108.1 Detailed Description

<<***eXtension***>> *(*p. *174)* Various settings that configure how DataReaders allocate and use physical memory for internal resources.

5.108.2 Function Documentation

5.108.2.1 static System::String ^ DDS::DataReaderResourceLimitsQosPolicy::get_datareaderresourcelimits_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::DataReaderResourceLimitsQosPolicy** (p. 486).

5.108.3 Properties

5.108.3.1 System::Int32 DDS::DataReaderResourceLimitsQosPolicy::AUTO_MAX_TOTAL_INSTANCES [static, get, inherited]

<<***eXtension***>> *(p. 174)* This value is used to make **DDS::DataReaderResourceLimitsQosPolicy::max_total_instances** (p. 494) equal to **DDS::ResourceLimitsQosPolicy::max_instances** (p. 1112).

5.109 DATA_WRITER_RESOURCE_LIMITS

<<*eXtension*>> *(p. 174)* Various settings that configure how a **DDS::DataWriter** (p. 499) allocates and uses physical memory for internal resources.

Classes

ˆ struct **DDS::DataWriterResourceLimitsQosPolicy**

*Various settings that configure how a **DDS::DataWriter** (p. 499) allocates and uses physical memory for internal resources.*

Enumerations

ˆ 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.

Functions

ˆ static System::String^ **DDS::DataWriterResourceLimitsQosPolicy::get_-datawriterresourcelimits_qos_policy_name** ()

*Stringified human-readable name for **DDS::DataWriterResourceLimitsQosPolicy** (p. 552).*

5.109.1 Detailed Description

<<eXtension>> (p. 174) Various settings that configure how a **DDS::DataWriter** (p. 499) allocates and uses physical memory for internal resources.

5.109.2 Enumeration Type Documentation

5.109.2.1 enum DDS::DataWriterResourceLimitsInstanceReplacementKind

Sets the kinds of instances that can be replaced when instance resource limits are reached.

When **DDS::ResourceLimitsQosPolicy::max_instances** (p. 1112) is reached, a **DDS::DataWriter** (p. 499) 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. 555).

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. 499) will replace the oldest instance satisfying that kind. For example, when the kind is DDS::DataWriterResourceLimitsInstanceReplacementKind::UNREGISTERED_INSTANCE_REPLACEMENT, a **DDS::DataWriter** (p. 499) 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. 499) will replace the old instance with the new instance, and invoke, if available, the DDS::DataWriterListener_InstanceReplacedCallback to notify the user about an instance being replaced.

A **DDS::DataWriter** (p. 499) checks for replaceable instances in the following order, stopping once a replaceable instance is found:

If **DDS::DataWriterResourceLimitsQosPolicy::replace_empty_instances** (p. 556) is true, a **DDS::DataWriter** (p. 499) first tries replacing instances that have no samples. These empty instances can be unregistered, disposed, or alive. Next, a **DDS::DataWriter** (p. 499) tries replacing unregistered instances. Since an unregistered instance indicates that the **DDS::DataWriter** (p. 499) is done modifying it, unregistered instances are replaced before instances of any other state (alive, disposed). This is the same as the DDS::DataWriterResourceLimitsInstanceReplacementKind::UNREGISTERED_

INSTANCE_REPLACEMENT kind. Then, a **DDS::DataWriter** (p. 499) tries replacing what is specified by **DDS::DataWriterResourceLimitsQosPolicy::instance_replacement** (p. 555). 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. 552)

Enumerator:

UNREGISTERED_INSTANCE_REPLACEMENT Allows a **DDS::DataWriter** (p. 499) 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. 555) **[default]**

ALIVE_INSTANCE_REPLACEMENT Allows a **DDS::DataWriter** (p. 499) to reclaim alive acknowledged instances.

When an unregistered acknowledged instance is not available to reclaim, this kind allows a **DDS::DataWriter** (p. 499) 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.

DISPOSED_INSTANCE_REPLACEMENT Allows a **DDS::DataWriter** (p. 499) to reclaim disposed acknowledged instances.

When an unregistered acknowledged instance is not available to reclaim, this kind allows a **DDS::DataWriter** (p. 499) to reclaim a disposed acknowledged instance. The least recently disposed instance will be reclaimed.

ALIVE_THEN_DISPOSED_INSTANCE_REPLACEMENT Allows a **DDS::DataWriter** (p. 499) 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. 499) first try reclaiming an alive acknowledged instance. If no instance is reclaimable, then it trys reclaiming a disposed acknowledged instance. The least recently used (i.e. registered, written, or disposed) instance will be reclaimed.

DISPOSED_THEN_ALIVE_INSTANCE_REPLACEMENT
Allows a **DDS::DataWriter** (p. 499) 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. 499) first try reclaiming a disposed acknowledged instance. If no instance is reclaimable, then it trys reclaiming an alive acknowledged instance. The least recently used (i.e. disposed, registered, or written) instance will be reclaimed.

ALIVE_OR_DISPOSED_INSTANCE_REPLACEMENT Allows a **DDS::DataWriter** (p. 499) to reclaim a either an alive acknowledged instance or a disposed acknowledged instance.

When an unregistered acknowledged instance is not available to reclaim, this kind allows a **DDS::DataWriter** (p. 499) to reclaim either an alive acknowledged instance or a disposed acknowledged instance. If both instance kinds are available to reclaim, the **DDS::DataWriter** (p. 499) will reclaim the least recently used (i.e. disposed, registered, or written) instance.

5.109.3 Function Documentation

5.109.3.1 static System::String ^ DDS::DataWriterResourceLimitsQosPolicy::get_datawriterresourcelimits_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::DataWriterResourceLimitsQosPolicy** (p. 552).

5.110 DATA_READER_PROTOCOL

<<***eXtension***>> *(*p. *174)* Specifies the DataReader-specific protocol QoS.

Classes

- struct **DDS::DataReaderProtocolQosPolicy**

  Along with ***DDS::WireProtocolQosPolicy*** *(*p. *1423)* *and* ***DDS::DataWriterProtocolQosPolicy*** *(*p. *529)**, this QoS policy configures the DDS on-the-network protocol (RTPS).*

Functions

- static System::String^ **DDS::DataReaderProtocolQosPolicy::get_datareaderprotocol_qos_policy_name** ()

  Stringified human-readable name for ***DDS::DataReaderProtocolQosPolicy*** *(*p. *465)**.*

5.110.1 Detailed Description

<<***eXtension***>> *(*p. *174)* Specifies the DataReader-specific protocol QoS.

5.110.2 Function Documentation

5.110.2.1 static System::String ^ DDS::DataReaderProtocolQosPolicy::get_datareaderprotocol_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::DataReaderProtocolQosPolicy** (p. 465).

5.111 DATA_WRITER_PROTOCOL

<<*eXtension*>> *(p. 174)* Along with **DDS::WireProtocolQosPolicy** (p. 1423) and **DDS::DataReaderProtocolQosPolicy** (p. 465), this QoS policy configures the DDS on-the-network protocol (RTPS).

Classes

- struct **DDS::DataWriterProtocolQosPolicy**

  *Protocol that applies only to **DDS::DataWriter** (p. 499) instances.*

Functions

- static System::String^ **DDS::DataWriterProtocolQosPolicy::get_datawriterprotocol_qos_policy_name** ()

  *Stringified human-readable name for **DDS::DataWriterProtocolQosPolicy** (p. 529).*

5.111.1 Detailed Description

<<*eXtension*>> *(p. 174)* Along with **DDS::WireProtocolQosPolicy** (p. 1423) and **DDS::DataReaderProtocolQosPolicy** (p. 465), this QoS policy configures the DDS on-the-network protocol (RTPS).

5.111.2 Function Documentation

5.111.2.1 static System::String ^ DDS::DataWriterProtocolQosPolicy::get_datawriterprotocol_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::DataWriterProtocolQosPolicy** (p. 529).

5.112 SYSTEM_RESOURCE_LIMITS

<<***eXtension***>> *(p. 174)* Configures DomainParticipant-independent resources used by RTI Data Distribution Service.

Classes

- struct **DDS::SystemResourceLimitsQosPolicy**

  *Configures **DDS::DomainParticipant** (p. 577)-independent resources used by RTI Data Distribution Service. Mainly used to change the maximum number of **DDS::DomainParticipant** (p. 577) entities that can be created within a single process (address space).*

Functions

- static System::String^ **DDS::SystemResourceLimitsQosPolicy::get_systemresourcelimits_qos_policy_name** ()

  *Stringified human-readable name for **DDS::SystemResourceLimitsQosPolicy** (p. 1247).*

5.112.1 Detailed Description

<<***eXtension***>> *(p. 174)* Configures DomainParticipant-independent resources used by RTI Data Distribution Service.

5.112.2 Function Documentation

5.112.2.1 static System::String ^ DDS::SystemResourceLimitsQosPolicy::get_systemresourcelimits_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::SystemResourceLimitsQosPolicy** (p. 1247).

5.113 DOMAIN_PARTICIPANT_-RESOURCE_LIMITS

<<eXtension>> *(p. 174)* Various settings that configure how a **DDS::DomainParticipant** (p. 577) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.

Classes

- struct **DDS::AllocationSettings_t**

  Resource allocation settings.

- struct **DDS::DomainParticipantResourceLimitsQosPolicy**

  *Various settings that configure how a **DDS::DomainParticipant** (p. 577) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.*

Functions

- static System::String^ **DDS::DomainParticipantResourceLimitsQosPolicy::get_-domainparticipantresourcelimits_qos_policy_name** ()

  *Stringified human-readable name for **DDS::DomainParticipantResourceLimitsQosPolicy** (p. 688).*

5.113.1 Detailed Description

<<eXtension>> *(p. 174)* Various settings that configure how a **DDS::DomainParticipant** (p. 577) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.

5.113.2 Function Documentation

5.113.2.1 static System::String ^ DDS::DomainParticipantResourceLimitsQosPolicy::get_-domainparticipantresourcelimits_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::DomainParticipantResourceLimitsQosPolicy** (p. 688).

5.114 EVENT

<<*eXtension*>> *(p. 174)* Configures the internal thread in a **DomainParticipant** (p. 577) that handles timed events.

Classes

^ class **DDS::EventQosPolicy**

Settings for event.

Functions

^ static System::String^ **DDS::EventQosPolicy::get_event_qos_policy_name** ()

*Stringified human-readable name for **DDS::EventQosPolicy** (p. 858).*

5.114.1 Detailed Description

<<*eXtension*>> *(p. 174)* Configures the internal thread in a **DomainParticipant** (p. 577) that handles timed events.

5.114.2 Function Documentation

5.114.2.1 static System::String ^ DDS::EventQosPolicy::get_event_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::EventQosPolicy** (p. 858).

5.115 DATABASE

<<*eXtension*>> *(p. 174)* Various threads and resource limits settings used by RTI Data Distribution Service to control its internal database.

Classes

- class **DDS::DatabaseQosPolicy**

  Various threads and resource limits settings used by RTI Data Distribution Service to control its internal database.

Functions

- static System::String^ **DDS::DatabaseQosPolicy::get_database_qos_policy_name** ()

  *Stringified human-readable name for **DDS::DatabaseQosPolicy** (p. 428).*

5.115.1 Detailed Description

<<*eXtension*>> *(p. 174)* Various threads and resource limits settings used by RTI Data Distribution Service to control its internal database.

5.115.2 Function Documentation

5.115.2.1 static System::String ^ DDS::DatabaseQosPolicy::get_database_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::DatabaseQosPolicy** (p. 428).

5.116 RECEIVER_POOL

*<<**eXtension**>> (p. 174)* Configures threads used by RTI Data Distribution Service to receive and process data from transports (for example, UDP sockets).

Classes

- class **DDS::ReceiverPoolQosPolicy**

  Configures threads used by RTI Data Distribution Service to receive and process data from transports (for example, UDP sockets).

Functions

- static System::String^ **DDS::ReceiverPoolQosPolicy::get_receiverpool_qos_policy_name** ()

  *Stringified human-readable name for **DDS::ReceiverPoolQosPolicy** (p. 1090).*

5.116.1 Detailed Description

*<<**eXtension**>> (p. 174)* Configures threads used by RTI Data Distribution Service to receive and process data from transports (for example, UDP sockets).

5.116.2 Function Documentation

5.116.2.1 static System::String ^ DDS::ReceiverPoolQosPolicy::get_receiverpool_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::ReceiverPoolQosPolicy** (p. 1090).

5.117 PUBLISH_MODE

<<***eXtension***>> *(*p. *174)* Specifies how RTI Data Distribution Service sends application data on the network. This QoS policy can be used to tell RTI Data Distribution Service to use its *own* thread to send data, instead of the user thread.

Classes

ˆ class **DDS::PublishModeQosPolicy**

Specifies how RTI Data Distribution Service sends application data on the network. This QoS policy can be used to tell RTI Data Distribution Service to use its own *thread to send data, instead of the user thread.*

Enumerations

ˆ enum **DDS::PublishModeQosPolicyKind** {

DDS::SYNCHRONOUS_PUBLISH_MODE_QOS,

DDS::ASYNCHRONOUS_PUBLISH_MODE_QOS }

Kinds of publishing mode.

Functions

ˆ static System::String^ **DDS::PublishModeQosPolicy::get_publishmode_qos_policy_name** ()

Stringified human-readable name for ***DDS::PublishModeQosPolicy*** *(*p. *1077).*

5.117.1 Detailed Description

<<***eXtension***>> *(*p. *174)* Specifies how RTI Data Distribution Service sends application data on the network. This QoS policy can be used to tell RTI Data Distribution Service to use its *own* thread to send data, instead of the user thread.

5.117.2 Enumeration Type Documentation

5.117.2.1 enum DDS::PublishModeQosPolicyKind

Kinds of publishing mode.

QoS:

DDS::PublishModeQosPolicy (p. 1077)

Enumerator:

SYNCHRONOUS_PUBLISH_MODE_QOS Indicates to send data synchronously.

If **DDS::DataWriterProtocolQosPolicy::push_on_write** (p. 532) is true, data is sent immediately in the context of **DDS::TypedDataWriter::write** (p. 1376).

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. 532)

[**default**] for **DDS::DataWriter** (p. 499)

ASYNCHRONOUS_PUBLISH_MODE_QOS Indicates to send data asynchronously.

Configures the **DDS::DataWriter** (p. 499) to delegate the task of data transmission to a separate publishing thread. The **DDS::TypedDataWriter::write** (p. 1376) call does not send the data, but instead schedules the data to be sent later by its associated **DDS::Publisher** (p. 1044).

Each **DDS::Publisher** (p. 1044) uses its dedicated publishing thread (**DDS::PublisherQos::asynchronous_publisher** (p. 1075)) to send data for all its asynchronous DataWriters. For each asynchronous **DataWriter** (p. 499), the associated **DDS::FlowController** (p. 867) determines when the publishing thread is allowed to send the data.

DDS::DataWriter::wait_for_asynchronous_publishing (p. 512) and **DDS::Publisher::wait_for_asynchronous_publishing** (p. 1063) enable you to determine when the data has actually been sent.

Note: **DDS::DataWriterProtocolQosPolicy::push_on_write** (p. 532) must be TRUE for Asynchronous DataWriters. Otherwise, samples will never be sent.

See also:

DDS::FlowController (p. 867)

DDS::HistoryQosPolicy (p. 898)
DDS::DataWriter::wait_for_asynchronous_publishing (p. 512)
DDS::Publisher::wait_for_asynchronous_publishing (p. 1063)
DDS::Transport_Property_t::gather_send_buffer_count_max

5.117.3 Function Documentation

5.117.3.1 static System::String ^ DDS::PublishModeQosPolicy::get_publishmode_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::PublishModeQosPolicy** (p. 1077).

5.118 DISCOVERY_CONFIG

<<***eXtension***>> *(*p. *174)* Specifies the discovery configuration QoS.

Classes

- struct **DDS::BuiltinTopicReaderResourceLimits_t**

  Built-in topic reader's resource limits.

- class **DDS::DiscoveryConfigQosPolicy**

  Settings for discovery configuration.

Enumerations

- 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.

Functions

- static System::String^ **DDS::DiscoveryConfigQosPolicy::get_discoveryconfig_qos_policy_name** ()

  Stringified human-readable name for ***DDS::DiscoveryConfigQosPolicy*** *(*p. *563)*.

5.118.1 Detailed Description

<<***eXtension***>> *(*p. *174)* Specifies the discovery configuration QoS.

5.118.2 Enumeration Type Documentation

5.118.2.1 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. 566)

Enumerator:

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. 565) QoS field. It maintains that contract by writing data to those other participants with a writer that has a DDS::LivelinessQosPolicyKind of DDS::LivelinessQosPolicyKind::AUTOMATIC_LIVELINESS_QOS or DDS::LivelinessQosPolicyKind::MANUAL_BY_PARTICIPANT_LIVELINESS_QOS and by asserting itself (at the **DDS::DiscoveryConfigQosPolicy::participant_liveliness_assert_period** (p. 565)) over the Simple Discovery Protocol.

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.

5.118.3 Function Documentation

5.118.3.1 static System::String ^ DDS::DiscoveryConfigQosPolicy::get_discoveryconfig_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::DiscoveryConfigQosPolicy** (p. 563).

5.119 TYPESUPPORT

<<*eXtension*>> *(p. 174)* Allows you to attach application-specific values to a **DataWriter** (p. 499) or **DataReader** (p. 433) that are passed to the serialization or deserialization routine of the associated data type.

Classes

- struct **DDS::TypeSupportQosPolicy**

  *Allows you to attach application-specific values to a **DataWriter** (p. 499) or **DataReader** (p. 433) that are passed to the serialization or deserialization routine of the associated data type.*

Functions

- static System::String^ **DDS::TypeSupportQosPolicy::get_typesupport_qos_policy_name** ()

  *Stringified human-readable name for **DDS::TypeSupportQosPolicy** (p. 1386).*

5.119.1 Detailed Description

<<*eXtension*>> *(p. 174)* Allows you to attach application-specific values to a **DataWriter** (p. 499) or **DataReader** (p. 433) that are passed to the serialization or deserialization routine of the associated data type.

5.119.2 Function Documentation

5.119.2.1 static System::String ^ DDS::TypeSupportQosPolicy::get_typesupport_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::TypeSupportQosPolicy** (p. 1386).

5.120 ASYNCHRONOUS_PUBLISHER

<<***eXtension***>> *(p. 174)* Specifies the asynchronous publishing settings of the **DDS::Publisher** (p. 1044) instances.

Classes

- class **DDS::AsynchronousPublisherQosPolicy**

  Configures the mechanism that sends user data in an external middleware thread.

Functions

- static System::String^ **DDS::AsynchronousPublisherQosPolicy::get_asynchronouspublisher_qos_policy_name** ()

  Stringified human-readable name for ***DDS::AsynchronousPublisherQosPolicy*** *(p. 371).*

5.120.1 Detailed Description

<<***eXtension***>> *(p. 174)* Specifies the asynchronous publishing settings of the **DDS::Publisher** (p. 1044) instances.

5.120.2 Function Documentation

5.120.2.1 static System::String ^ DDS::AsynchronousPublisherQosPolicy::get_asynchronouspublisher_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::AsynchronousPublisherQosPolicy** (p. 371).

5.121 EXCLUSIVE_AREA

*<<**eXtension**>> (p. 174)* Configures multi-thread concurrency and deadlock prevention capabilities.

Classes

- ^ struct **DDS::ExclusiveAreaQosPolicy**

  Configures multi-thread concurrency and deadlock prevention capabilities.

Functions

- ^ static System::String^ **DDS::ExclusiveAreaQosPolicy::get_exclusivearea_qos_policy_name** ()

  *Stringified human-readable name for **DDS::ExclusiveAreaQosPolicy** (p. 862).*

5.121.1 Detailed Description

*<<**eXtension**>> (p. 174)* Configures multi-thread concurrency and deadlock prevention capabilities.

5.121.2 Function Documentation

5.121.2.1 static System::String ^ DDS::ExclusiveAreaQosPolicy::get_exclusivearea_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::ExclusiveAreaQosPolicy** (p. 862).

5.122 BATCH

<<***eXtension***>> *(p. 174)* Batch QoS policy used to enable batching in **DDS::DataWriter** (p. 499) instances.

Classes

- struct **DDS::BatchQosPolicy**

  Used to configure batching of multiple samples into a single network packet in order to increase throughput for small samples.

Functions

- static System::String^ **DDS::BatchQosPolicy::get_batch_qos_policy_name** ()

  *Stringified human-readable name for **DDS::BatchQosPolicy** (p. 376).*

5.122.1 Detailed Description

<<***eXtension***>> *(p. 174)* Batch QoS policy used to enable batching in **DDS::DataWriter** (p. 499) instances.

5.122.2 Function Documentation

5.122.2.1 static System::String ^ DDS::BatchQosPolicy::get_batch_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::BatchQosPolicy** (p. 376).

5.123 LOCATORFILTER

*<<**eXtension**>>* *(p. 174)* The QoS policy used to report the configuration of a MultiChannel **DataWriter** (p. 499) as part of **DDS::PublicationBuiltinTopicData** (p. 1030).

Classes

- class **DDS::LocatorFilter_t**

  *The QoS policy used to report the configuration of a MultiChannel **DataWriter** (p. 499) as part of **DDS::PublicationBuiltinTopicData** (p. 1030).*

- class **DDS::LocatorFilterSeq**

  Declares IDL `sequence<` ***DDS::LocatorFilter_t*** *(p. 970)* `>`.

- class **DDS::LocatorFilterQosPolicy**

  *The QoS policy used to report the configuration of a MultiChannel **DataWriter** (p. 499) as part of **DDS::PublicationBuiltinTopicData** (p. 1030).*

Functions

- static System::String^ **DDS::LocatorFilterQosPolicy::get_locator_filter_qos_policy_name** ()

  *Stringified human-readable name for **DDS::LocatorFilterQosPolicy** (p. 972).*

5.123.1 Detailed Description

*<<**eXtension**>>* *(p. 174)* The QoS policy used to report the configuration of a MultiChannel **DataWriter** (p. 499) as part of **DDS::PublicationBuiltinTopicData** (p. 1030).

5.123.2 Function Documentation

5.123.2.1 static System::String ^ DDS::LocatorFilterQosPolicy::get_locator_filter_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::LocatorFilterQosPolicy** (p. 972).

5.124 MULTICHANNEL

*<<**eXtension**>> (p. 174)* Configures the ability of a **DataWriter** (p. 499) to send data on different multicast groups (addresses) based on the value of the data.

Classes

- class **DDS::ChannelSettings_t**

  Type used to configure the properties of a channel.

- class **DDS::ChannelSettingsSeq**

  Declares IDL `sequence<` ***DDS::ChannelSettings_t*** *(p. 403)* `>`.

- class **DDS::MultiChannelQosPolicy**

  *Configures the ability of a **DataWriter** (p. 499) to send data on different multicast groups (addresses) based on the value of the data.*

Functions

- static System::String^ **DDS::MultiChannelQosPolicy::get_multichannel_qos_policy_name** ()

  *Stringified human-readable name for **DDS::MultiChannelQosPolicy** (p. 981).*

5.124.1 Detailed Description

*<<**eXtension**>> (p. 174)* Configures the ability of a **DataWriter** (p. 499) to send data on different multicast groups (addresses) based on the value of the data.

5.124.2 Function Documentation

5.124.2.1 static System::String ^ DDS::MultiChannelQosPolicy::get_multichannel_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::MultiChannelQosPolicy** (p. 981).

5.125 PROPERTY

<<***eXtension***>> *(*p. *174)* Stores name/value (string) pairs that can be used to configure certain parameters of RTI Data Distribution Service that are not exposed through formal QoS policies. Can also be used to store and propagate application-specific name/value pairs that can be retrieved by user code during discovery.

Classes

- class **DDS::Property_t**

  Properties are name/value pairs objects.

- class **DDS::PropertySeq**

  Declares IDL `sequence` < ***DDS::Property_t*** *(*p. *1022)* >.

- class **DDS::PropertyQosPolicy**

  Stores name/value(string) pairs that can be used to configure certain parameters of RTI Data Distribution Service that are not exposed through formal QoS policies. Can also be used to store and propagate application-specific name/value pairs that can be retrieved by user code during discovery.

- class **DDS::PropertyQosPolicyHelper**

  Policy Helpers which facilitate management of the properties in the input policy.

Functions

- static System::String^ **DDS::PropertyQosPolicy::get_property_qos_policy_name** ()

  Stringified human-readable name for ***DDS::PropertyQosPolicy*** *(*p. *1023).*

- static Int32 **DDS::PropertyQosPolicyHelper::get_number_of_properties** (**PropertyQosPolicy**^ policy)

  Gets the number of properties in the input policy.

- static void **DDS::PropertyQosPolicyHelper::assert_property** (**PropertyQosPolicy**^ policy, String^ name, String^ value, System::Boolean propagate)

  Asserts the property identified by name in the input policy.

- static void **DDS::PropertyQosPolicyHelper::add_property** (**PropertyQosPolicy**^ policy, String^ name, String^ value, System::Boolean propagate)

  Adds a new property to the input policy.

- static **Property_t**^ **DDS::PropertyQosPolicyHelper::lookup_property** (**PropertyQosPolicy**^ policy, System::String^ name)

  Searches for a property in the input policy given its name.

- static void **DDS::PropertyQosPolicyHelper::remove_property** (**PropertyQosPolicy**^ policy, String^ name)

  Removes a property from the input policy.

- static void **DDS::PropertyQosPolicyHelper::get_properties** (**PropertyQosPolicy**^ policy, **PropertySeq**^ properties, String^ name_prefix)

  Retrieves a list of properties whose names match the input prefix.

5.125.1 Detailed Description

<<***eXtension***>> *(*p. *174)* Stores name/value (string) pairs that can be used to configure certain parameters of RTI Data Distribution Service 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.

See **DDS::PropertyQosPolicy** (p. 1023)

5.125.2 Function Documentation

5.125.2.1 static System::String ^ DDS::PropertyQosPolicy::get_property_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::PropertyQosPolicy** (p. 1023).

5.125.2.2 static Int32 DDS::PropertyQosPolicyHelper::get_number_of_properties (PropertyQosPolicy^ *policy*) [static, inherited]

Gets the number of properties in the input policy.

Precondition:

policy cannot be null.

Parameters:

policy *<<in>>* *(p. 175)* Input policy.

Returns:

Number of properties.

5.125.2.3 static void DDS::PropertyQosPolicyHelper::assert_property (PropertyQosPolicy^ *policy*, String^ *name*, String^ *value*, System::Boolean *propagate*) [static, inherited]

Asserts the property identified by name in the input policy.

If the property already exists, this function replaces its current value with the new one.

If the property identified by name does not exist, this function adds it to the property set.

This function increases the maximum number of elements of the policy sequence when this number is not enough to store the new property.

Precondition:

policy, name and value cannot be null.

Parameters:

policy *<<in>>* *(p. 175)* Input policy.

name *<<in>>* *(p. 175)* Property name.

value *<<in>>* *(p. 175)* Property value.

propagate *<<in>>* *(p. 175)* Indicates if the property will be propagated on discovery.

Returns:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_OutOfResources** (p. 1122).

5.125.2.4 static void DDS::PropertyQosPolicyHelper::add_property (PropertyQosPolicy^ *policy*, String^ *name*, String^ *value*, System::Boolean *propagate*) [static, inherited]

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 Data Distribution Service.

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_PreconditionNotMet** (p. 1123).

Precondition:

policy, name and value cannot be null.
The property is not in the policy.

Parameters:

policy *<<in>>* *(p. 175)* Input policy.

name *<<in>>* *(p. 175)* Property name.

value *<<in>>* *(p. 175)* Property value.

propagate *<<in>>* *(p. 175)* Indicates if the property will be propagated on discovery.

Returns:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_OutOfResources** (p. 1122) or **DDS::Retcode_PreconditionNotMet** (p. 1123)

5.125.2.5 static Property_t ^ DDS::PropertyQosPolicyHelper::lookup_property (PropertyQosPolicy^ *policy*, System::String^ *name*) [static, inherited]

Searches for a property in the input policy given its name.

Precondition:

policy, name and value cannot be null.

Parameters:

policy *<<in>>* *(p. 175)* Input policy.

name *<<in>>* *(p. 175)* Property name.

Returns:

On success, the function returns the first property with the given name. Otherwise, the function returns NULL.

5.125.2.6 static void DDS::PropertyQosPolicyHelper::remove_property (PropertyQosPolicy^ *policy*, String^ *name*) [static, inherited]

Removes a property from the input policy.

If the property does not exist, the function fails with **DDS::Retcode_PreconditionNotMet** (p. 1123).

Precondition:

policy and name cannot be null.
The property is in the policy.

Parameters:

policy *<<in>>* *(p. 175)* Input policy.

name *<<in>>* *(p. 175)* Property name.

Returns:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_PreconditionNotMet** (p. 1123).

5.125.2.7 static void DDS::PropertyQosPolicyHelper::get_properties (PropertyQosPolicy^ *policy*, PropertySeq^ *properties*, String^ *name_prefix*) [static, inherited]

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_OutOfResources** (p. 1122).

Precondition:

policy, properties and name_prefix cannot be null.

Parameters:

policy *<<in>>* *(p. 175)* Input policy.

properties *<<inout>>* *(p. 176)* A **DDS::PropertySeq** (p. 1027) object where the set or list of properties will be returned.

name_prefix Name prefix.

Returns:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_OutOfResources** (p. 1122).

5.126 Entity Support

DDS::Entity (p. 845), **DDS::Listener** (p. 952) and related items.

Classes

- ˆ class **DDS::Listener**

  <<**interface**>> (p. 175) *Abstract base class for all* ***Listener*** *(p. 952) interfaces.*

- ˆ class **DDS::Entity**

  <<**interface**>> (p. 175) *Abstract base class for all the DDS objects that support QoS policies, a listener, and a status condition.*

- ˆ class **DDS::DomainEntity**

  <<**interface**>> (p. 175) *Abstract base class for all DDS entities except for the* ***DDS::DomainParticipant*** *(p. 577).*

5.126.1 Detailed Description

DDS::Entity (p. 845), **DDS::Listener** (p. 952) and related items.

DDS::Entity (p. 845) subtypes are created and destroyed by factory objects. With the exception of **DDS::DomainParticipant** (p. 577), whose factory is **DDS::DomainParticipantFactory** (p. 649), all **DDS::Entity** (p. 845) factory objects are themselves **DDS::Entity** (p. 845) subtypes as well.

Important: all **DDS::Entity** (p. 845) delete operations are inherently thread-unsafe. The user must take extreme care that a given **DDS::Entity** (p. 845) 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. 845) is undefined.

5.127 ENTITY_NAME

<<*eXtension*>> *(p. 174)* Assigns a name to a **DDS::DomainParticipant** (p. 577). This name will be visible during the discovery process and in RTI tools to help you visualize and debug your system.

Classes

- class **DDS::EntityNameQosPolicy**

  *Assigns a name to a **DDS::DomainParticipant** (p. 577). This name will be visible during the discovery process and in RTI tools to help you visualize and debug your system.*

Functions

- static System::String^ **DDS::EntityNameQosPolicy::get_entityname_qos_policy_name** ()

  *Stringified human-readable name for **DDS::EntityNameQosPolicy** (p. 854).*

5.127.1 Detailed Description

<<*eXtension*>> *(p. 174)* Assigns a name to a **DDS::DomainParticipant** (p. 577). This name will be visible during the discovery process and in RTI tools to help you visualize and debug your system.

5.127.2 Function Documentation

5.127.2.1 static System::String ^ DDS::EntityNameQosPolicy::get_entityname_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::EntityNameQosPolicy** (p. 854).

5.128 PROFILE

<<***eXtension***>> *(p. 174)* Configures the way that XML documents containing QoS profiles are loaded by RTI Data Distribution Service.

Classes

- class **DDS::ProfileQosPolicy**

  Configures the way that XML documents containing QoS profiles are loaded by RTI Data Distribution Service.

Functions

- static System::String^ **DDS::ProfileQosPolicy::get_profile_qos_policy_name** ()

  Stringified human-readable name for ***DDS::ProfileQosPolicy*** *(p. 1019).*

5.128.1 Detailed Description

<<***eXtension***>> *(p. 174)* Configures the way that XML documents containing QoS profiles are loaded by RTI Data Distribution Service.

5.128.2 Function Documentation

5.128.2.1 static System::String ^ DDS::ProfileQosPolicy::get_profile_qos_policy_name () [inline, static, inherited]

Stringified human-readable name for **DDS::ProfileQosPolicy** (p. 1019).

5.129 Conditions and WaitSets

DDS::Condition (p. 408) and **DDS::WaitSet** (p. 1411) and related items.

Classes

- struct **DDS::WaitSetProperty_t**

  <<**eXtension**>> (p. 174) *Specifies the* ***DDS::WaitSet*** *(p. 1411) behavior for multiple trigger events.*

- class **DDS::Condition**

  <<**interface**>> (p. 175) *Root class for all the conditions that may be attached to a* ***DDS::WaitSet*** *(p. 1411).*

- class **DDS::ConditionSeq**

  Instantiates `DDS::Sequence` (p. 1163) *<* ***DDS::Condition*** *(p. 408) >.*

- class **DDS::GuardCondition**

  <<**interface**>> (p. 175) *A specific* ***DDS::Condition*** *(p. 408) whose* `trigger_value` *is completely under the control of the application.*

- class **DDS::StatusCondition**

  <<**interface**>> (p. 175) *A specific* ***DDS::Condition*** *(p. 408) that is associated with each* ***DDS::Entity*** *(p. 845).*

- class **DDS::WaitSet**

  <<**interface**>> (p. 175) *Allows an application to wait until one or more of the attached* ***DDS::Condition*** *(p. 408) objects has a* `trigger_value` *of true or else until the timeout expires.*

5.129.1 Detailed Description

DDS::Condition (p. 408) and **DDS::WaitSet** (p. 1411) and related items.

5.130 Sequence Support

The **DDS::Sequence** (p. 1163) interface allows you to work with variable-length collections of homogeneous data.

Modules

- **Built-in Sequences**

  Defines sequences of primitive data type.

Classes

- class **DDS::Sequence< T >**

  <<**interface**>> (p. 175) <<**generic**>> (p. 175) *A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as* `Foo` (p. 877).

- class **DDS::LoanableSequence< E >**

  A sequence implementation used internally by the middleware to efficiently manage memory during ***DDS::TypedDataReader::read*** *(p. 1341) and* ***DDS::TypedDataReader::take*** *(p. 1342) operations.*

- class **FooSeq**

  <<**interface**>> (p. 175) <<**generic**>> (p. 175) *A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as* `Foo` (p. 877).

5.130.1 Detailed Description

The **DDS::Sequence** (p. 1163) 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. 109) are pre-defined instantiations for the primitive data types.

When you use the **rtiddsgen** (p. 196) code generation tool, it will automatically generate concrete sequence instantiations for each of your own custom types.

Chapter 6

Class Documentation

6.1 DDS::AllocationSettings_t Struct Reference

Resource allocation settings.

`#include <managed_infrastructure.h>`

Public Attributes

- System::Int32 **initial_count**

  The initial count of resources.

- System::Int32 **max_count**

  The maximum count of resources.

- System::Int32 **incremental_count**

  The incremental count of resources.

6.1.1 Detailed Description

Resource allocation settings.

QoS:

DDS::DomainParticipantResourceLimitsQosPolicy (p. 688)

6.1.2 Member Data Documentation

6.1.2.1 System::Int32 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)

6.1.2.2 System::Int32 DDS::AllocationSettings_t::max_count

The maximum count of resources.

The maximum resources to be allocated.

[default] It depends on the case.

[range] [1, 1 million] or DDS::LENGTH_UNLIMITED, > initial_count or (= initial_count only if increment_count == 0)

6.1.2.3 System::Int32 DDS::AllocationSettings_t::incremental_count

The incremental count of resources.

The resource to be allocated when more resources are needed.

[default] It depends on the case.

[range] -1 or [1,1 million] or (= 0 only if initial_count == max_count)

6.2 DDS::AsynchronousPublisherQosPolicy Class Reference

Configures the mechanism that sends user data in an external middleware thread.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_asynchronouspublisher_qos_policy_name** ()

  *Stringified human-readable name for **DDS::AsynchronousPublisherQosPolicy*** *(p. 371).*

Public Attributes

- **ThreadSettings_t^ thread**

  Settings of the publishing thread.

- **ThreadSettings_t^ asynchronous_batch_thread**

  Settings of the batch flushing thread.

Properties

- System::Boolean **disable_asynchronous_write** `[get, set]`

  Disable asynchronous publishing.

- System::Boolean **disable_asynchronous_batch** `[get, set]`

  Disable asynchronous batch flushing.

6.2.1 Detailed Description

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. 1044) instances.

The QoS policy specifies whether asynchronous publishing and asynchronous batch flushing are enabled for the **DDS::DataWriter** (p. 499) entities belonging to this **DDS::Publisher** (p. 1044). If so, the publisher will spawn up to two threads, one for asynchronous publishing and one for asynchronous batch flushing.

See also:

DDS::BatchQosPolicy (p. 376).
DDS::PublishModeQosPolicy (p. 1077).

Entity:

DDS::Publisher (p. 1044)

Properties:

RxO (p. 268) = N/A

Changeable (p. 269) = **NO** (p. 269)

6.2.2 Usage

You can use this QoS policy to reduce the amount of time your application thread spends sending data.

You can also use it, along with **DDS::PublishModeQosPolicy** (p. 1077) and a **DDS::FlowController** (p. 867), to send large data reliably. "Large" in this context means that the data that cannot be sent as a single packet by a network transport. For example, to send data larger than 63K reliably using UDP/IP, you must configure RTI Data Distribution Service to fragment the data and send it asynchronously.

The asynchronous *publisher* thread is shared by all DDS::PublishModeQosPolicyKind::ASYNCHRONOUS_PUBLISH_MODE_QOS **DDS::DataWriter** (p. 499) instances that belong to this publisher and handles their data transmission chores.

The asynchronous *batch flushing* thread is shared by all **DDS::DataWriter** (p. 499) 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. 499) entities, the instances must belong to different **DDS::Publisher** (p. 1044) instances.

A **DDS::Publisher** (p. 1044) must have asynchronous publishing enabled for its **DDS::DataWriter** (p. 499) instances to write asynchronously.

A **DDS::Publisher** (p. 1044) must have asynchronous batch flushing enabled in order to flush the batches of its **DDS::DataWriter** (p. 499) instances asynchronously. However, no asynchronous batch flushing thread will be started until the first **DDS::DataWriter** (p. 499) instance with batching enabled is created from this **DDS::Publisher** (p. 1044).

6.2.3 Member Data Documentation

6.2.3.1 ThreadSettings_t ^ DDS::AsynchronousPublisherQosPolicy::thread

Settings of the publishing thread.

There is only one asynchronous publishing thread per **DDS::Publisher** (p. 1044).

[**default**] priority below normal.

The actual value depends on your architecture:

For Windows: -2

For Solaris: OS default priority

For Linux: OS default priority

For LynxOS: 13

For Integrity: 80

For VxWorks: 110

For all others: OS default priority.

[**default**] The actual value depends on your architecture:

For Windows: OS default stack size

For Solaris: OS default stack size

For Linux: OS default stack size

For LynxOS: 4∗16∗1024

For Integrity: 4∗20∗1024

For VxWorks: 4∗16∗1024

For all others: OS default stack size.

[**default**] mask = DDS::ThreadSettingsKind::THREAD_SETTINGS_KIND_MASK_DEFAULT

6.2.3.2 ThreadSettings_t ^ DDS::AsynchronousPublisherQosPolicy::asynchronous_batch_thread

Settings of the batch flushing thread.

There is only one asynchronous batch flushing thread per **DDS::Publisher** (p. 1044).

[**default**] priority below normal.

The actual value depends on your architecture:

For Windows: -2

For Solaris: OS default priority

For Linux: OS default priority

For LynxOS: 13

For Integrity: 80

For VxWorks: 110

For all others: OS default priority.

[**default**] The actual value depends on your architecture:

For Windows: OS default stack size

For Solaris: OS default stack size

For Linux: OS default stack size

For LynxOS: 4∗16∗1024

For Integrity: 4∗20∗1024

For VxWorks: 4∗16∗1024

For all others: OS default stack size.

[**default**] mask = DDS::ThreadSettingsKind::THREAD_SETTINGS_KIND_MASK_DEFAULT

6.2.4 Property Documentation

6.2.4.1 System:: Boolean DDS::AsynchronousPublisherQosPolicy::disable_asynchronous_write [get, set]

Disable asynchronous publishing.

If set to true, any **DDS::DataWriter** (p. 499) created with

DDS::PublishModeQosPolicyKind::ASYNCHRONOUS_PUBLISH_MODE_QOS will fail with **DDS::Retcode_InconsistentPolicy** (p. 1119).

[**default**] false

6.2.4.2 System:: Boolean DDS::AsynchronousPublisherQosPolicy::disable_asynchronous_batch [get, set]

Disable asynchronous batch flushing.

If set to true, any **DDS::DataWriter** (p. 499) created with batching enabled will fail with **DDS::Retcode_InconsistentPolicy** (p. 1119).

If **DDS::BatchQosPolicy::max_flush_delay** (p. 378) is different than **DDS::Duration_t::DURATION_INFINITE** (p. 253), **DDS::AsynchronousPublisherQosPolicy::disable_asynchronous_batch** (p. 375) must be set false.

[**default**] false

6.3 DDS::BatchQosPolicy Struct Reference

Used to configure batching of multiple samples into a single network packet in order to increase throughput for small samples.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_batch_qos_policy_name** ()

  *Stringified human-readable name for **DDS::BatchQosPolicy*** (p. 376).

Public Attributes

- System::Int32 **max_data_bytes**

  The maximum cumulative length of all serialized samples in a batch.

- System::Int32 **max_samples**

  The maximum number of samples in a batch.

- **Duration_t max_flush_delay**

  The maximum flush delay.

- **Duration_t source_timestamp_resolution**

  Batch source timestamp resolution.

Properties

- System::Boolean **enable** `[get, set]`

  Specifies whether or not batching is enabled.

- System::Boolean **thread_safe_write** `[get, set]`

  Determines whether or not the write operation is thread safe.

6.3.1 Detailed Description

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. 499)

Properties:

RxO (p. 268) = NO
Changeable (p. 269) = **UNTIL ENABLE** (p. 269)

6.3.2 Member Data Documentation

6.3.2.1 System::Int32 DDS::BatchQosPolicy::max_data_bytes

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 Data Distribution Service will send the sample in a single batch.

[**default**] 1024

[**range**] [1,DDS::LENGTH_UNLIMITED]

6.3.3 Consistency

The setting of **DDS::BatchQosPolicy::max_data_bytes** (p. 377) must be consistent with **DDS::BatchQosPolicy::max_samples** (p. 378). For these two values to be consistent, they cannot be both DDS::LENGTH_UNLIMITED.

6.3.3.1 System::Int32 DDS::BatchQosPolicy::max_samples

The maximum number of samples in a batch.

A batch is flushed automatically when this maximum is reached.

[default] DDS::LENGTH_UNLIMITED

[range] [1,DDS::LENGTH_UNLIMITED]

6.3.4 Consistency

The setting of **DDS::BatchQosPolicy::max_samples** (p. 378) must be consistent with **DDS::BatchQosPolicy::max_data_bytes** (p. 377). For these two values to be consistent, they cannot be both DDS::LENGTH_UNLIMITED.

6.3.4.1 Duration_t DDS::BatchQosPolicy::max_flush_delay

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_t::DURATION_INFINITE** (p. 253)

[range] [0,**DDS::Duration_t::DURATION_INFINITE** (p. 253)]

6.3.5 Consistency

The setting of **DDS::BatchQosPolicy::max_flush_delay** (p. 378) must be consistent with **DDS::AsynchronousPublisherQosPolicy::disable_asynchronous_batch** (p. 375) and **DDS::BatchQosPolicy::thread_safe_write** (p. 379). If the delay is different than **DDS::Duration_t::DURATION_INFINITE** (p. 253), **DDS::AsynchronousPublisherQosPolicy::disable_asynchronous_batch** (p. 375) must be set to false and **DDS::BatchQosPolicy::thread_safe_write** (p. 379) must be set to true.

6.3.5.1 Duration_t DDS::BatchQosPolicy::source_timestamp_resolution

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_t::DURATION_INFINITE** (p. 253), 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_t::DURATION_INFINITE** (p. 253).

[**default**] **DDS::Duration_t::DURATION_INFINITE** (p. 253)

[**range**] [0,**DDS::Duration_t::DURATION_INFINITE** (p. 253)]

6.3.6 Consistency

The setting of **DDS::BatchQosPolicy::source_timestamp_resolution** (p. 378) must be consistent with **DDS::BatchQosPolicy::thread_safe_write** (p. 379). If **DDS::BatchQosPolicy::thread_safe_write** (p. 379) is set to false, **DDS::BatchQosPolicy::source_timestamp_resolution** (p. 378) must be set to **DDS::Duration_t::DURATION_INFINITE** (p. 253).

6.3.7 Property Documentation

6.3.7.1 System:: Boolean DDS::BatchQosPolicy::enable [get, set]

Specifies whether or not batching is enabled.

[**default**] false

6.3.7.2 System:: Boolean DDS::BatchQosPolicy::thread_safe_write [get, set]

Determines whether or not the write operation is thread safe.

If this parameter is set to true, multiple threads can call write on the **DDS::DataWriter** (p. 499) concurrently.

[**default**] true

6.3.8 Consistency

The setting of **DDS::BatchQosPolicy::thread_safe_write** (p. 379) must be consistent with **DDS::BatchQosPolicy::source_timestamp_resolution** (p. 378). If **DDS::BatchQosPolicy::thread_safe_write** (p. 379) is set to false, **DDS::BatchQosPolicy::source_timestamp_resolution** (p. 378) must be set to **DDS::Duration_t::DURATION_INFINITE** (p. 253).

6.4 DDS::BooleanSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < System::Boolean >.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::BooleanSeq::

Public Member Functions

- ^ **BooleanSeq** ()
  Constructs an empty sequence of booleans with an initial maximum of zero.
- ^ **BooleanSeq** (System::Int32 max)
  Constructs an empty sequence of booleans with the given initial maximum.
- ^ **BooleanSeq** (**BooleanSeq**^ booleans)
  Constructs a new sequence containing the given booleans.

6.4.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < System::Boolean >.

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

System::Boolean
DDS::Sequence (p. 1163)

6.4.2 Constructor & Destructor Documentation

6.4.2.1 DDS::BooleanSeq::BooleanSeq () `[inline]`

Constructs an empty sequence of booleans with an initial maximum of zero.

6.4.2.2 DDS::BooleanSeq::BooleanSeq (System::Int32 *max*) `[inline]`

Constructs an empty sequence of booleans with the given initial maximum.

6.4.2.3 DDS::BooleanSeq::BooleanSeq (BooleanSeq^ *booleans*) [inline]

Constructs a new sequence containing the given booleans.

Parameters:

booleans the initial contents of this sequence

6.5 DDS::BuiltinTopicKey_t Struct Reference

The key type of the built-in topic types.

`#include <managed_infrastructure.h>`

Public Attributes

- System::Int32 **value1**

  *An array of four integers that uniquely represents a remote **DDS::Entity** (p. 845).*

- System::Int32 **value2**

  *An array of four integers that uniquely represents a remote **DDS::Entity** (p. 845).*

- System::Int32 **value3**

  *An array of four integers that uniquely represents a remote **DDS::Entity** (p. 845).*

- System::Int32 **value4**

  *An array of four integers that uniquely represents a remote **DDS::Entity** (p. 845).*

6.5.1 Detailed Description

The key type of the built-in topic types.

Each remote **DDS::Entity** (p. 845) to be discovered is can be uniquely identified by this key. This is the key of all the built-in topic data types.

See also:

DDS::ParticipantBuiltinTopicData (p. 1002)
DDS::TopicBuiltinTopicData (p. 1268)
DDS::PublicationBuiltinTopicData (p. 1030)
DDS::SubscriptionBuiltinTopicData (p. 1233)

6.5.2 Member Data Documentation

6.5.2.1 System::Int32 DDS::BuiltinTopicKey_t::value1

An array of four integers that uniquely represents a remote **DDS::Entity** (p. 845).

6.5.2.2 System::Int32 DDS::BuiltinTopicKey_t::value2

An array of four integers that uniquely represents a remote **DDS::Entity** (p. 845).

6.5.2.3 System::Int32 DDS::BuiltinTopicKey_t::value3

An array of four integers that uniquely represents a remote **DDS::Entity** (p. 845).

6.5.2.4 System::Int32 DDS::BuiltinTopicKey_t::value4

An array of four integers that uniquely represents a remote **DDS::Entity** (p. 845).

6.6 DDS::BuiltinTopicReaderResourceLimits_t Struct Reference

Built-in topic reader's resource limits.

`#include <managed_infrastructure.h>`

Public Attributes

- System::Int32 **initial_samples**

  Initial number of samples.

- System::Int32 **max_samples**

  Maximum number of samples.

- System::Int32 **initial_infos**

  Initial number of sample infos.

- System::Int32 **max_infos**

  Maximum number of sample infos.

- System::Int32 **initial_outstanding_reads**

  The initial number of outstanding reads that have not call finish yet on the same built-in topic ***DDS::DataReader*** *(*p. 433*).*

- System::Int32 **max_outstanding_reads**

  The maximum number of outstanding reads that have not called finish yet on the same built-in topic ***DDS::DataReader*** (p. 433*).*

- System::Int32 **max_samples_per_read**

  Maximum number of samples that can be read/taken on a same built-in topic ***DDS::DataReader*** *(*p. 433*).*

6.6.1 Detailed Description

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 Data Distribution Service assigns a unique key to each entity in a domain.

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)

QoS:

DDS::DiscoveryConfigQosPolicy (p. 563)

6.6.2 Member Data Documentation

6.6.2.1 System::Int32 DDS::BuiltinTopicReaderResourceLimits_-t::initial_samples

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

6.6.2.2 System::Int32 DDS::BuiltinTopicReaderResourceLimits_-t::max_samples

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

[range] [1, 1 million] or DDS::LENGTH_UNLIMITED, >= initial_samples

6.6.2.3 System::Int32 DDS::BuiltinTopicReaderResourceLimits_-t::initial_infos

Initial number of sample infos.

The initial number of info units that a built-in topic **DDS::DataReader** (p. 433) can have. Info units are used to store **DDS::SampleInfo** (p. 1148).

[default] 64

[range] [1, 1 million] <= max_infos

6.6.2.4 System::Int32 DDS::BuiltinTopicReaderResourceLimits_t::max_infos

Maximum number of sample infos.

The maximum number of info units that a built-in topic **DDS::DataReader** (p. 433) can use to store **DDS::SampleInfo** (p. 1148).

[default] DDS::LENGTH_UNLIMITED

[range] [1, 1 million] or DDS::LENGTH_UNLIMITED, >= initial_infos

6.6.2.5 System::Int32 DDS::BuiltinTopicReaderResourceLimits_t::initial_outstanding_reads

The initial number of outstanding reads that have not call finish yet on the same built-in topic **DDS::DataReader** (p. 433).

[default] 2

[range] [1, 65536] or DDS::LENGTH_UNLIMITED <= max_outstanding_reads

6.6.2.6 System::Int32 DDS::BuiltinTopicReaderResourceLimits_t::max_outstanding_reads

The maximum number of outstanding reads that have not called finish yet on the same built-in topic **DDS::DataReader** (p. 433).

[default] DDS::LENGTH_UNLIMITED

[range] [1, 65536] or DDS::LENGTH_UNLIMITED, >= initial_outstanding_reads

6.6.2.7 System::Int32 DDS::BuiltinTopicReaderResourceLimits_t::max_samples_per_read

Maximum number of samples that can be read/taken on a same built-in topic **DDS::DataReader** (p. 433).

[default] 1024

[range] [1, 65536]

6.7 DDS::Bytes Struct Reference

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

`#include <managed_bytes.h>`

Inheritance diagram for DDS::Bytes::

Public Member Functions

- **Bytes** ()
  Default Constructor.
- **Bytes** (System::Int32 size)
  Constructor that specifies the size of the allocated bytes array.
- virtual System::Boolean **copy_from** (**Bytes**^ src)
  Copy src into this object.

Public Attributes

- System::Int32 **length**
  Number of bytes to serialize.
- System::Int32 **offset**
  Offset from which to start serializing bytes .
- array< System::Byte >^ **value**
  DDS::Bytes *(p. 388) array value.*

6.7.1 Detailed Description

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

6.7.2 Constructor & Destructor Documentation

6.7.2.1 DDS::Bytes::Bytes ()

Default Constructor.

The default constructor initializes the newly created object with null value, zero length, and zero offset.

6.7.2.2 DDS::Bytes::Bytes (System::Int32 *size*)

Constructor that specifies the size of the allocated bytes array.

After this method is called, length and offset are set to zero.

Parameters:

size <<***in***>> *(*p. 175*)* Size of the allocated bytes array.

6.7.3 Member Function Documentation

6.7.3.1 virtual System::Boolean DDS::Bytes::copy_from (Bytes^ *src*) [virtual]

Copy src into this object.

This method performs a deep copy of `src` and it allocates memory for the value if required.

Parameters:

src <<***in***>> *(*p. 175*)* Object to copy from.

Returns:

true if success. Otherwise, false.

Exceptions:

ArgumentNullException if `src` is null.

6.7.4 Member Data Documentation

6.7.4.1 System::Int32 DDS::Bytes::length

Number of bytes to serialize.

6.7.4.2 System::Int32 DDS::Bytes::offset

Offset from which to start serializing bytes .

The first position of the bytes array has offset 0.

6.7.4.3 array<System::Byte> ^ DDS::Bytes::value

DDS::Bytes (p. 388) array value.

6.8 DDS::BytesDataReader Class Reference

<<***interface***>> *(p. 175)* Instantiates `DataReader` (p. 433) < **DDS::Bytes** (p. 388) >.

`#include <managed_bytesSupport.h>`

Inheritance diagram for DDS::BytesDataReader::

6.8.1 Detailed Description

<<***interface***>> *(p. 175)* Instantiates `DataReader` (p. 433) < **DDS::Bytes** (p. 388) >.

See also:

DDS::TypedDataReader (p. 1338)
DDS::DataReader (p. 433)

6.9 DDS::BytesDataWriter Class Reference

<<***interface***>> *(p. 175)* Instantiates `DataWriter` (p. 499) < **DDS::Bytes** (p. 388) >.

`#include <managed_bytesSupport.h>`

Inheritance diagram for DDS::BytesDataWriter::

Public Member Functions

- ^ void **write** (array< System::Byte >^octets, System::Int32 offset, System::Int32 length, **DDS::InstanceHandle_t**% handle)

  <<**eXtension**>> (p. 174) *Modifies the value of a* ***DDS::Bytes*** *(p. 388) data instance.*

- ^ void **write** (**ByteSeq**^ octets, **DDS::InstanceHandle_t**% handle)

  <<**eXtension**>> (p. 174) *Modifies the value of a* ***DDS::Bytes*** *(p. 388) data instance.*

- ^ void **write_w_timestamp** (array< System::Byte >^octets, System::Int32 offset, System::Int32 length, **DDS::InstanceHandle_t**% handle, **DDS::Time_t**% source_timestamp)

  <<**eXtension**>> (p. 174) *Performs the same function as* ***DDS::BytesDataWriter::write*** *(p. 393) except that it also provides the value for the* `source_timestamp`.

- ^ void **write_w_timestamp** (**ByteSeq**^ octets, **DDS::InstanceHandle_t**% handle, **DDS::Time_t**% source_timestamp)

  <<**eXtension**>> (p. 174) *Performs the same function as* ***DDS::BytesDataWriter::write*** *(p. 393) except that it also provides the value for the* `source_timestamp`.

6.9.1 Detailed Description

<<***interface***>> *(p. 175)* Instantiates `DataWriter` (p. 499) < **DDS::Bytes** (p. 388) >.

See also:

DDS::TypedDataWriter (p. 1368)
DDS::DataWriter (p. 499)

6.9.2 Member Function Documentation

6.9.2.1 void DDS::BytesDataWriter::write (array< System::Byte >^ *octets*, System::Int32 *offset*, System::Int32 *length*, DDS::InstanceHandle_t% *handle*)

<<eXtension>> (p. 174) Modifies the value of a **DDS::Bytes** (p. 388) data instance.

Parameters:

octets *<<in>> (p. 175)* Array of bytes to be published.

offset *<<in>> (p. 175)* Offset from which to start publishing.

length *<<in>> (p. 175)* Number of bytes to be published.

handle *<<in>> (p. 175)* The special value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) should be used always.

See also:

DDS::TypedDataWriter::write (p. 1376)

6.9.2.2 void DDS::BytesDataWriter::write (ByteSeq^ *octets*, DDS::InstanceHandle_t% *handle*)

<<eXtension>> (p. 174) Modifies the value of a **DDS::Bytes** (p. 388) data instance.

Parameters:

octets *<<in>> (p. 175)* **Sequence** (p. 1163) of bytes to be published.

handle *<<in>> (p. 175)* The special value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) should be used always.

See also:

DDS::TypedDataWriter::write (p. 1376)

6.9.2.3 void DDS::BytesDataWriter::write_w_timestamp (array< System::Byte >^ *octets*, System::Int32 *offset*, System::Int32 *length*, DDS::InstanceHandle_t% *handle*, DDS::Time_t% *source_timestamp*)

<<eXtension>> (p. 174) Performs the same function as **DDS::BytesDataWriter::write** (p. 393) except that it also provides the value for the `source_timestamp`.

Parameters:

octets *<<in>> (p. 175)* Array of bytes to be published.

offset *<<in>> (p. 175)* Offset from which to start publishing.

length *<<in>> (p. 175)* Number of bytes to be published.

handle *<<in>> (p. 175)* The special value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) should be used always.

source_timestamp *<<in>> (p. 175)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation. See **DDS::TypedDataWriter::write_w_timestamp** (p. 1378). Cannot be NULL.

See also:

DDS::TypedDataWriter::write_w_timestamp (p. 1378)

6.9.2.4 void DDS::BytesDataWriter::write_w_timestamp (ByteSeq^ *octets*, DDS::InstanceHandle_t% *handle*, DDS::Time_t% *source_timestamp*)

<<eXtension>> (p. 174) Performs the same function as **DDS::BytesDataWriter::write** (p. 393) except that it also provides the value for the `source_timestamp`.

Parameters:

octets *<<in>> (p. 175)* **Sequence** (p. 1163) of bytes to be published.

handle *<<in>> (p. 175)* The special value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) should be used always.

source_timestamp *<<in>> (p. 175)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation. See **DDS::TypedDataWriter::write_w_timestamp** (p. 1378). Cannot be NULL.

See also:

DDS::TypedDataWriter::write_w_timestamp (p. 1378)

6.10 DDS::ByteSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < System::Byte >.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::ByteSeq::

Public Member Functions

- ^ **ByteSeq** ()

  Constructs an empty sequence of bytes with an initial maximum of zero.

- ^ **ByteSeq** (System::Int32 max)

  Constructs an empty sequence of bytes with the given initial maximum.

- ^ **ByteSeq** (**ByteSeq**^ bytes)

  Construct a new sequence containing the given bytes.

6.10.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < System::Byte >.

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

System::Byte
DDS::Sequence (p. 1163)

6.10.2 Constructor & Destructor Documentation

6.10.2.1 DDS::ByteSeq::ByteSeq () `[inline]`

Constructs an empty sequence of bytes with an initial maximum of zero.

6.10.2.2 DDS::ByteSeq::ByteSeq (System::Int32 *max*) `[inline]`

Constructs an empty sequence of bytes with the given initial maximum.

6.10.2.3 DDS::ByteSeq::ByteSeq (ByteSeq^ *bytes*) [inline]

Construct a new sequence containing the given bytes.

Parameters:

bytes the initial contents of this sequence

6.11 DDS::BytesSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < **DDS::Bytes** (p. 388) > .

`#include <managed_bytes.h>`

Public Member Functions

- ^ **BytesSeq** ()

  Constructs an empty sequence of ***DDS::Bytes*** *(p. 388) objects with an initial maximum of zero.*

- ^ **BytesSeq** (System::Int32 initialMaximum)

  Constructs an empty sequence of ***DDS::Bytes*** *(p. 388) objects with the given initial maximum.*

- ^ **BytesSeq** (**BytesSeq**^ src)

  Copy constructor.

6.11.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < **DDS::Bytes** (p. 388) > .

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

DDS::Bytes (p. 388)

6.11.2 Constructor & Destructor Documentation

6.11.2.1 DDS::BytesSeq::BytesSeq () `[inline]`

Constructs an empty sequence of **DDS::Bytes** (p. 388) objects with an initial maximum of zero.

6.11.2.2 DDS::BytesSeq::BytesSeq (System::Int32 *initialMaximum*) `[inline]`

Constructs an empty sequence of **DDS::Bytes** (p. 388) objects with the given initial maximum.

6.11.2.3 DDS::BytesSeq::BytesSeq (BytesSeq^ *src*) [inline]

Copy constructor.

6.12 DDS::BytesTypeSupport Class Reference

<<***interface***>> *(p. 175)* **DDS::Bytes** (p. 388) type support.

`#include <managed_bytesSupport.h>`

Inherits DDS::TypedTypeSupport< T >.

Static Public Member Functions

- ^ static System::String^ **get_type_name** ()

  Get the default name for the ***DDS::Bytes*** *(p. 388) type.*

- ^ static void **print_data** (**Bytes**^ a_data)

  <<**eXtension**>> (p. 174) *Print value of data type to standard out.*

- ^ static void **register_type** (**DDS::DomainParticipant**^ participant, System::String^ type_name)

  Allows an application to communicate to RTI Data Distribution Service the existence of the ***DDS::Bytes*** *(p. 388) data type.*

- ^ static void **unregister_type** (**DDS::DomainParticipant**^ participant, System::String^ type_name)

  Allows an application to unregister the ***DDS::Bytes*** *(p. 388) data type from RTI Data Distribution Service. After calling unregister_type, no further communication using this type is possible.*

6.12.1 Detailed Description

<<***interface***>> *(p. 175)* **DDS::Bytes** (p. 388) type support.

6.12.2 Member Function Documentation

6.12.2.1 static System::String ^ DDS::BytesTypeSupport::get_type_name () [static]

Get the default name for the **DDS::Bytes** (p. 388) type.

Can be used for calling **DDS::BytesTypeSupport::register_type** (p. 400) or creating **DDS::Topic** (p. 1258).

Returns:

default name for the **DDS::Bytes** (p. 388) type.

See also:

DDS::BytesTypeSupport::register_type (p. 400)
DDS::DomainParticipant::create_topic (p. 621)

6.12.2.2 static void DDS::BytesTypeSupport::print_data (Bytes^ *a_data*) [static]

*<<**eXtension**>> (p. 174)* Print value of data type to standard out.

The *generated* implementation of the operation knows how to print value of a data type.

Parameters:

a_data *<<**in**>> (p. 175)* **DDS::Bytes** (p. 388) to be printed.

6.12.2.3 static void DDS::BytesTypeSupport::register_type (DDS::DomainParticipant^ *participant*, System::String^ *type_name*) [static]

Allows an application to communicate to RTI Data Distribution Service the existence of the **DDS::Bytes** (p. 388) data type.

By default, The **DDS::Bytes** (p. 388) built-in type is automatically registered when a **DomainParticipant** (p. 577) is created using the type_name returned by **DDS::BytesTypeSupport::get_type_name** (p. 399). Therefore, the usage of this function is optional and it is only required when the automatic built-in type registration is disabled using the participant property "dds.builtin_type.auto_register".

This method can also be used to register the same **DDS::BytesTypeSupport** (p. 399) with a **DDS::DomainParticipant** (p. 577) using different values for the type_name.

If `register_type` is called multiple times with the same **DDS::DomainParticipant** (p. 577) and `type_name`, the second (and subsequent) registrations are ignored by the operation.

Parameters:

participant *<<**in**>> (p. 175)* the **DDS::DomainParticipant** (p. 577) to register the data type **DDS::Bytes** (p. 388) with. Cannot be null.

type_name *<<**in**>> (p. 175)* the type name under with the data type **DDS::Bytes** (p. 388) is registered with the participant; this type name is used when creating a new **DDS::Topic** (p. 1258). (See

DDS::DomainParticipant::create_topic (p. 621).) The name may not be null or longer than 255 characters.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_PreconditionNotMet** (p. 1123) or **DDS::Retcode_OutOfResources** (p. 1122).

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. 621)

6.12.2.4 static void DDS::BytesTypeSupport::unregister_type (DDS::DomainParticipant^ *participant*, System::String^ *type_name*) [static]

Allows an application to unregister the **DDS::Bytes** (p. 388) data type from RTI Data Distribution Service. After calling unregister_type, no further communication using this type is possible.

Precondition:

The **DDS::Bytes** (p. 388) type with `type_name` is registered with the participant and all **DDS::Topic** (p. 1258) objects referencing the type have been destroyed. If the type is not registered with the participant, or if any **DDS::Topic** (p. 1258) is associated with the type, the operation will fail with **DDS::Retcode_Error** (p. 1116).

Postcondition:

All information about the type is removed from RTI Data Distribution Service. No further communication using this type is possible.

Parameters:

participant *<<in>>* *(p. 175)* the **DDS::DomainParticipant** (p. 577) to unregister the data type **DDS::Bytes** (p. 388) from. Cannot be null.

type_name *<<in>>* *(p. 175)* the type name under with the data type **DDS::Bytes** (p. 388) is registered with the participant. The name should match a name that has been previously used to register a type with the participant. Cannot be null.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_BadParameter** (p. 1115) or **DDS::Retcode_Error** (p. 1116)

MT Safety:

SAFE.

See also:

DDS::BytesTypeSupport::register_type (p. 400)

6.13 DDS::ChannelSettings_t Class Reference

Type used to configure the properties of a channel.

`#include <managed_infrastructure.h>`

Public Attributes

- ^ **TransportMulticastSettingsSeq^ multicast_settings**

  *A sequence of **DDS::TransportMulticastSettings_t** (p. 1289) used to configure the multicast addresses associated with a channel.*

- ^ System::String^ **filter_expression**

  A logical expression used to determine the data that will be published in the channel.

6.13.1 Detailed Description

Type used to configure the properties of a channel.

QoS:

DDS::MultiChannelQosPolicy (p. 981)

6.13.2 Member Data Documentation

6.13.2.1 TransportMulticastSettingsSeq ^ DDS::ChannelSettings_t::multicast_settings

A sequence of **DDS::TransportMulticastSettings_t** (p. 1289) 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 four (A locator is defined by a transport alias, a multicast address and a port)

[**default**] Empty sequence (invalid value)

6.13.2.2 System::String ^ DDS::ChannelSettings_t::filter_expression

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. 983)

The filter expression length (including NULL-terminated character) cannot be greater than **DDS::DomainParticipantResourceLimitsQosPolicy::channel_filter_expression_max_length** (p. 704).

See also:

Queries and Filters Syntax (p. 184)

[**default**] NULL (invalid value)

6.14 DDS::ChannelSettingsSeq Class Reference

Declares IDL `sequence<` **DDS::ChannelSettings_t** (p. 403) `>`.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::ChannelSettingsSeq::

6.14.1 Detailed Description

Declares IDL `sequence<` **DDS::ChannelSettings_t** (p. 403) `>`.

A sequence of **DDS::ChannelSettings_t** (p. 403) used to configure the channels' properties. If the length of the sequence is zero, the **DDS::MultiChannelQosPolicy** (p. 981) has no effect.

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

DDS::ChannelSettings_t (p. 403)

6.15 DDS::CharSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < System::Char >.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::CharSeq::

Public Member Functions

- ^ **CharSeq** ()

  Constructs an empty sequence of single-byte (serialized) characters with an initial maximum of zero.

- ^ **CharSeq** (System::Int32 max)

  Constructs an empty sequence of single-byte (serialized) characters with the given initial maximum.

- ^ **CharSeq** (**CharSeq**^ chars)

  Constructs a new sequence containing the given single-byte (serialized) characters.

6.15.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < System::Char >.

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

System::Char
DDS::Sequence (p. 1163)

6.15.2 Constructor & Destructor Documentation

6.15.2.1 DDS::CharSeq::CharSeq () `[inline]`

Constructs an empty sequence of single-byte (serialized) characters with an initial maximum of zero.

6.15.2.2 DDS::CharSeq::CharSeq (System::Int32 *max*) [inline]

Constructs an empty sequence of single-byte (serialized) characters with the given initial maximum.

6.15.2.3 DDS::CharSeq::CharSeq (CharSeq^ *chars*) [inline]

Constructs a new sequence containing the given single-byte (serialized) characters.

Parameters:

chars the initial contents of this sequence

6.16 DDS::Condition Class Reference

<<***interface***>> *(*p. *175)* Root class for all the conditions that may be attached to a **DDS::WaitSet** (p. 1411).

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::Condition::

Public Member Functions

- virtual System::Boolean **get_trigger_value** () override

  Retrieve the `trigger_value`.

6.16.1 Detailed Description

<<***interface***>> *(*p. *175)* Root class for all the conditions that may be attached to a **DDS::WaitSet** (p. 1411).

This basic class is specialised in three classes:

DDS::GuardCondition (p. 892), **DDS::StatusCondition** (p. 1183), and **DDS::ReadCondition** (p. 1084).

A **DDS::Condition** (p. 408) has a `trigger_value` that can be true or false and is set automatically by RTI Data Distribution Service.

See also:

DDS::WaitSet (p. 1411)

6.16.2 Member Function Documentation

6.16.2.1 virtual System::Boolean DDS::Condition::get_trigger_value () [pure virtual]

Retrieve the `trigger_value`.

Returns:

the trigger value.

Implemented in **DDS::GuardCondition** (p. 893), **DDS::StatusCondition** (p. 1185), **DDS::ReadCondition** (p. 1085), and **DDS::QueryCondition** (p. 1083).

6.17 DDS::ConditionSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < **DDS::Condition** (p. 408) >.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::ConditionSeq::

6.17.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < **DDS::Condition** (p. 408) >.

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

DDS::WaitSet (p. 1411)
DDS::Sequence (p. 1163)

6.18 NDDS::Config_LibraryVersion_t Struct Reference

The version of a single library shipped as part of an RTI Data Distribution Service distribution.

`#include <managed_config_dotnet.h>`

Public Attributes

- System::Int32 **major**

  The major version of a single RTI Data Distribution Service library.

- System::Int32 **minor**

  The minor version of a single RTI Data Distribution Service library.

- System::Byte **release**

  The release letter of a single RTI Data Distribution Service library.

- System::Int32 **build**

  The build number of a single RTI Data Distribution Service library.

6.18.1 Detailed Description

The version of a single library shipped as part of an RTI Data Distribution Service distribution.

RTI Data Distribution Service is comprised of a number of separate libraries. Although RTI Data Distribution Service 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.

6.18.2 Member Data Documentation

6.18.2.1 System::Int32 NDDS::Config_LibraryVersion_t::major

The major version of a single RTI Data Distribution Service library.

6.18.2.2 System::Int32 NDDS::Config_LibraryVersion_t::minor

The minor version of a single RTI Data Distribution Service library.

6.18.2.3 System::Byte NDDS::Config_LibraryVersion_t::release

The release letter of a single RTI Data Distribution Service library.

6.18.2.4 System::Int32 NDDS::Config_LibraryVersion_t::build

The build number of a single RTI Data Distribution Service library.

6.19 NDDS::ConfigLogger Class Reference

<<***interface***>> *(p. 175)* The singleton type used to configure RTI Data Distribution Service logging.

`#include <managed_config_dotnet.h>`

Public Member Functions

- ^ **LogVerbosity get_verbosity** ()

  Get the verbosity at which RTI Data Distribution Service is currently logging diagnostic information.

- ^ **LogVerbosity get_verbosity_by_category** (**LogCategory** category)

  Get the verbosity at which RTI Data Distribution Service is currently logging diagnostic information in the given category.

- ^ void **set_verbosity** (**LogVerbosity** verbosity)

  Set the verbosity at which RTI Data Distribution Service will log diagnostic information.

- ^ void **set_verbosity_by_category** (**LogCategory** category, **LogVerbosity** verbosity)

  Set the verbosity at which RTI Data Distribution Service will log diagnostic information in the given category.

- ^ **LogPrintFormat get_print_format** ()

  Get the file to which the logged output is redirected.

- ^ System::Boolean **set_print_format** (**LogPrintFormat** print_format)

  Set the message format that RTI Data Distribution Service will use to log diagnostic information.

Static Public Member Functions

- ^ static **ConfigLogger^ get_instance** ()

  Get the singleton instance of this type.

- ^ static void **set_log_delegate** (LogCallbackDelegate^ cb)

  Install a custom callback for logging.

^ static LogCallbackDelegate^ **get_log_delegate** ()

Get the custom logging callback, if any has been installed.

6.19.1 Detailed Description

<<***interface***>> *(p. 175)* The singleton type used to configure RTI Data Distribution Service logging.

6.19.2 Member Function Documentation

6.19.2.1 static ConfigLogger ^ NDDS::ConfigLogger::get_instance () [inline, static]

Get the singleton instance of this type.

6.19.2.2 LogVerbosity NDDS::ConfigLogger::get_verbosity ()

Get the verbosity at which RTI Data Distribution Service is currently logging diagnostic information.

The default verbosity if **NDDS::ConfigLogger::set_verbosity** (p. 415) is never called is NDDS::LogVerbosity::NDDS_CONFIG_LOG_VERBOSITY_ERROR.

If **NDDS::ConfigLogger::set_verbosity_by_category** (p. 415) has been used to set different verbosities for different categories of messages, this method will return the maximum verbosity of all categories.

6.19.2.3 LogVerbosity NDDS::ConfigLogger::get_verbosity_by_category (LogCategory *category*)

Get the verbosity at which RTI Data Distribution Service is currently logging diagnostic information in the given category.

The default verbosity if **NDDS::ConfigLogger::set_verbosity** (p. 415) and **NDDS::ConfigLogger::set_verbosity_by_category** (p. 415) are never called is NDDS::LogVerbosity::NDDS_CONFIG_LOG_VERBOSITY_ERROR.

6.19.2.4 void NDDS::ConfigLogger::set_verbosity (LogVerbosity *verbosity*)

Set the verbosity at which RTI Data Distribution Service will log diagnostic information.

Note: Logging at high verbosities will be detrimental to your application's performance. Your default setting should typically remain at NDDS::LogVerbosity::NDDS_CONFIG_LOG_VERBOSITY_WARNING or below. (The default verbosity if you never set it is NDDS::LogVerbosity::NDDS_CONFIG_LOG_VERBOSITY_ERROR.)

6.19.2.5 void NDDS::ConfigLogger::set_verbosity_by_category (LogCategory *category*, LogVerbosity *verbosity*)

Set the verbosity at which RTI Data Distribution Service will log diagnostic information in the given category.

6.19.2.6 LogPrintFormat NDDS::ConfigLogger::get_print_format ()

Get the file to which the logged output is redirected.

If no output file has been registered through NDDS::ConfigLogger::set_output_file, this method will return NULL. In this case, logged output will on most platforms go to standard out as if through printf. 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). Get the current message format that RTI Data Distribution Service is using to log diagnostic information. If **NDDS::ConfigLogger::set_print_format** (p. 415) is never called, the default format is NDDS::LogPrintFormat::NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT.

6.19.2.7 System::Boolean NDDS::ConfigLogger::set_print_format (LogPrintFormat *print_format*)

Set the message format that RTI Data Distribution Service will use to log diagnostic information.

If you use **NDDS::ConfigLogger::set_log_delegate()** (p. 416), you cannot also use **set_print_format()** (p. 415). You must use NDDS::LogPrintFormat::NDDS_CONFIG_LOG_PRINT_FORMAT_DEFAULT.

6.19.2.8 static void NDDS::ConfigLogger::set_log_delegate (LogCallbackDelegate^ *cb*) [static]

Install a custom callback for logging.

Custom log delegates are not compatible with the advanced logging information enabled by **NDDS::ConfigLogger::set_print_format** (p. 415).

See also:

NDDS::ConfigLogger::get_log_delegate (p. 416)
NDDS::LogCallbackDelegate

6.19.2.9 static LogCallbackDelegate ^ NDDS::ConfigLogger::get_log_delegate () [static]

Get the custom logging callback, if any has been installed.

See also:

NDDS::ConfigLogger::set_log_delegate (p. 416)
NDDS::LogCallbackDelegate

6.20 NDDS::ConfigVersion Class Reference

<<*interface*>> *(p. 175)* The version of an RTI Data Distribution Service distribution.

`#include <managed_config_dotnet.h>`

Public Member Functions

- **DDS::ProductVersion_t get_product_version** ()

  Get the RTI Data Distribution Service product version.

- **Config_LibraryVersion_t get_dotnet_api_version** ()

  Get the version of the .NET API library.

- **Config_LibraryVersion_t get_cpp_api_version** ()

  Get the version of the C++ API library.

- **Config_LibraryVersion_t get_c_api_version** ()

  Get the version of the C API library.

- **Config_LibraryVersion_t get_core_version** ()

  Get the version of the core library.

- System::String^ **to_string** ()

  Get this version in string form.

Static Public Member Functions

- static **ConfigVersion^ get_instance** ()

  Get the singleton instance of this type.

6.20.1 Detailed Description

<<*interface*>> *(p. 175)* The version of an RTI Data Distribution Service distribution.

The complete version is made up of the versions of the individual libraries that make up the product distribution.

6.20.2 Member Function Documentation

6.20.2.1 static ConfigVersion ^ NDDS::ConfigVersion::get_instance () [inline, static]

Get the singleton instance of this type.

6.20.2.2 DDS::ProductVersion_t NDDS::ConfigVersion::get_product_version ()

Get the RTI Data Distribution Service product version.

6.20.2.3 Config_LibraryVersion_t NDDS::ConfigVersion::get_dotnet_api_version ()

Get the version of the .NET API library.

6.20.2.4 Config_LibraryVersion_t NDDS::ConfigVersion::get_cpp_api_version ()

Get the version of the C++ API library.

6.20.2.5 Config_LibraryVersion_t NDDS::ConfigVersion::get_c_api_version ()

Get the version of the C API library.

6.20.2.6 Config_LibraryVersion_t NDDS::ConfigVersion::get_core_version ()

Get the version of the core library.

6.20.2.7 System::String ^ NDDS::ConfigVersion::to_string ()

Get this version in string form.

Combine all of the constituent library versions into a single string.

6.21 DDS::ContentFilteredTopic Class Reference

<<***interface***>> *(p. 175)* Specialization of DDS::TopicDescription that allows for content-based subscriptions.

`#include <managed_topic.h>`

Inheritance diagram for DDS::ContentFilteredTopic::

Public Member Functions

- ^ System::String^ **get_filter_expression** ()

  Get the `filter_expression`*.*

- ^ void **get_expression_parameters** (**StringSeq**^ parameters)

  Get the `expression_parameters`*.*

- ^ void **set_expression_parameters** (**StringSeq**^ parameters)

  Set the `expression_parameters`*.*

- ^ void **append_to_expression_parameter** (System::Int32 index, System::String^ val)

  <<**eXtension**>> (p. 174) *Appends a string term to the specified parameter string.*

- ^ void **remove_from_expression_parameter** (System::Int32 index, System::String^ val)

  <<**eXtension**>> (p. 174) *Removes a string term from the specified parameter string.*

- ^ **Topic**^ **get_related_topic** ()

  Get the `related_topic`*.*

- ^ virtual System::String^ **get_type_name** ()

  Get the associated `type_name`*.*

- ^ virtual System::String^ **get_name** ()

  Get the name used to create this DDS::TopicDescription .

- ^ virtual **DomainParticipant**^ **get_participant** ()

*Get the **DDS::DomainParticipant** (p. 577) to which the DDS::TopicDescription belongs.*

Static Public Member Functions

- static **ContentFilteredTopic**^ **narrow** (**ITopicDescription**^ topic_description)

  *Narrow the given DDS::TopicDescription pointer to a **DDS::ContentFilteredTopic** (p. 419) pointer.*

6.21.1 Detailed Description

<<***interface***>> (p. 175) Specialization of DDS::TopicDescription that allows for content-based subscriptions.

It describes a more sophisticated subscription that indicates a **DDS::DataReader** (p. 433) does not want to necessarily see all values of each instance published under the **DDS::Topic** (p. 1258). Rather, it wants to see only the values whose contents satisfy certain criteria. This class therefore can be used to request content-based subscriptions.

The selection of the content is done using the `filter_expression` with parameters `expression_parameters`.

- The `filter_expression` attribute is a string that specifies the criteria to select the data samples of interest. It is similar to the WHERE part of an SQL clause.

- The `expression_parameters` attribute is a sequence of strings that give values to the 'parameters' (i.e. "%n" tokens) in the `filter_expression`. The number of supplied parameters must fit with the requested values in the `filter_expression` (i.e. the number of n tokens).

Queries and Filters Syntax (p. 184) describes the syntax of `filter_expression` and `expression_parameters`.

Note on Content-Based Filtering and Sparse Value Types

If you are a user of the **Dynamic Data** (p. 73) API, you may define *sparse value types*; that is, types for which every data sample need not include a value for every field defined in the type. (See DDS::TCKind::TK_SPARSE and **DDS::TypeCodeFactory::create_sparse_tc** (p. 1336).) In order for a filter expression on a field to be well defined, that field must be present in the data sample. That means that you will only be able to perform a content-based filter

on fields that are marked as **DDS::TypeCode::KEY_MEMBER** (p. 66) or **DDS::TypeCode::NONKEY_REQUIRED_MEMBER** (p. 66).

6.21.2 Member Function Documentation

6.21.2.1 static ContentFilteredTopic ^ DDS::ContentFilteredTopic::narrow (ITopicDescription^ *topic_description*) [static]

Narrow the given DDS::TopicDescription pointer to a **DDS::ContentFilteredTopic** (p. 419) pointer.

Returns:

DDS::ContentFilteredTopic (p. 419) if this DDS::TopicDescription is a **DDS::ContentFilteredTopic** (p. 419). Otherwise, return NULL.

6.21.2.2 System::String ^ DDS::ContentFilteredTopic::get_filter_expression ()

Get the `filter_expression`.

Return the `filter_expression` associated with the **DDS::ContentFilteredTopic** (p. 419).

Returns:

the `filter_expression`.

6.21.2.3 void DDS::ContentFilteredTopic::get_expression_parameters (StringSeq^ *parameters*)

Get the `expression_parameters`.

Return the `expression_parameters` associated with the **DDS::ContentFilteredTopic** (p. 419). `expression_parameters` is either specified on the last successful call to **DDS::ContentFilteredTopic::set_expression_parameters** (p. 422) or, if that method is never called, the parameters specified when the **DDS::ContentFilteredTopic** (p. 419) was created.

Parameters:

parameters *<<inout>> (p. 176)* the filter expression parameters. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DomainParticipant::create_contentfilteredtopic (p. 625)
DDS::ContentFilteredTopic::set_expression_parameters (p. 422)

6.21.2.4 void DDS::ContentFilteredTopic::set_expression_parameters (StringSeq^ *parameters*)

Set the `expression_parameters`.

Change the `expression_parameters` associated with the **DDS::ContentFilteredTopic** (p. 419).

Parameters:

parameters *<<**in**>> (p. 175)* the filter expression parameters Cannot be NULL.. Length of sequence cannot be greater than 100.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.21.2.5 void DDS::ContentFilteredTopic::append_to_expression_parameter (System::Int32 *index*, System::String^ *val*)

*<<**eXtension**>> (p. 174)* Appends a string term to the specified parameter string.

Appends the input string to the end of the specified parameter string, separated by a comma. If the original parameter string is enclosed in quotation marks ("), the resultant string will also be enclosed in quotation marks.

This method can be used in expression parameters associated with MATCH operators in order to add a pattern to the match pattern list. For example, if the filter expression parameter value is:

'IBM'

Then append_to_expression_parameter(0, "MSFT") would generate the new value:

'IBM,MSFT'

Parameters:

index <<***in***>> *(*p. *175)* The index of the parameter string to be modified. The first index is index 0. When using the **DDS::DomainParticipant::STRINGMATCHFILTER_NAME** (p. 41) filter, `index` *must* be 0.

val <<***in***>> *(*p. *175)* The string term to be appended to the parameter string.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.21.2.6 void DDS::ContentFilteredTopic::remove_from_expression_parameter (System::Int32 *index*, System::String^ *val*)

<<***eXtension***>> *(*p. *174)* Removes a string term from the specified parameter string.

Removes the input string from the specified parameter string. To be found and removed, the input string must exist as a complete term, bounded by comma separators or the strong boundary. If the original parameter string is enclosed in quotation marks ("), the resultant string will also be enclosed in quotation marks. If the removed term was the last entry in the string, the result will be a string of empty quotation marks.

This method can be used in expression parameters associated with MATCH operators in order to remove a pattern from the match pattern list. For example, if the filter expression paremeter value is:

'IBM,MSFT'

Then remove_from_expression_parameter(0, "IBM") would generate the expression:

'MSFT'

Parameters:

index <<***in***>> *(*p. *175)* The index of the parameter string to be modified. The first index is index 0. When using the **DDS::DomainParticipant::STRINGMATCHFILTER_NAME** (p. 41) filter, `index` *must* be 0.

val <<***in***>> *(*p. *175)* The string term to be removed from the parameter string.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.21.2.7 Topic ^ DDS::ContentFilteredTopic::get_related_topic ()

Get the `related_topic`.

Return the **DDS::Topic** (p. 1258) specified when the **DDS::ContentFilteredTopic** (p. 419) was created.

Returns:

The **DDS::Topic** (p. 1258) assocated with the **DDS::ContentFilteredTopic** (p. 419).

6.21.2.8 virtual System::String ^ DDS::ContentFilteredTopic::get_type_name () [virtual]

Get the associated `type_name`.

The type name defines a locally unique type for the publication or the subscription.

The `type_name` corresponds to a unique string used to register a type via the **FooTypeSupport::register_type** (p. 885) method.

Thus, the `type_name` implies an association with a corresponding **DDS::TypeSupport** (p. 1385) and this DDS::TopicDescription.

Returns:

the type name. The returned type name is valid until the DDS::TopicDescription is deleted.

Postcondition:

The result is non-NULL.

See also:

DDS::TypeSupport (p. 1385), **FooTypeSupport** (p. 884)

Implements **DDS::ITopicDescription** (p. 913).

6.21.2.9 virtual System::String ^ DDS::ContentFilteredTopic::get_name () [virtual]

Get the name used to create this DDS::TopicDescription .

Returns:

the name used to create this DDS::TopicDescription. The returned topic name is valid until the DDS::TopicDescription is deleted.

Postcondition:

The result is non-NULL.

Implements **DDS::ITopicDescription** (p. 913).

6.21.2.10 virtual DomainParticipant ^ DDS::ContentFilteredTopic::get_participant () [virtual]

Get the **DDS::DomainParticipant** (p. 577) to which the DDS::TopicDescription belongs.

Returns:

The **DDS::DomainParticipant** (p. 577) to which the DDS::TopicDescription belongs.

Postcondition:

The result is non-NULL.

Implements **DDS::ITopicDescription** (p. 914).

6.22 DDS::ContentFilterProperty_t Class Reference

<<*eXtension*>> *(p. 174)* Type used to provide all the required information to enable content filtering.

`#include <managed_infrastructure.h>`

Public Attributes

- ^ System::String^ **content_filter_topic_name**

  Name of the Content-filtered ***Topic*** *(p. 1258) associated with the Reader.*

- ^ System::String^ **related_topic_name**

  Name of the ***Topic*** *(p. 1258) related to the Content-filtered* ***Topic*** *(p. 1258).*

- ^ System::String^ **filter_class_name**

  Identifies the filter class this filter belongs to. RTPS can support multiple filter classes (SQL, regular expressions, custom filters, etc).

- ^ System::String^ **filter_expression**

  The actual filter expression. Must be a valid expression for the filter class specified using filterClassName.

- ^ **StringSeq**^ **expression_parameters**

  Defines the value for each parameter in the filter expression.

6.22.1 Detailed Description

<<*eXtension*>> *(p. 174)* Type used to provide all the required information to enable content filtering.

6.22.2 Member Data Documentation

6.22.2.1 System::String ^ DDS::ContentFilterProperty_t::content_filter_topic_name

Name of the Content-filtered **Topic** (p. 1258) associated with the Reader.

6.22.2.2 System::String ^ DDS::ContentFilterProperty_t::related_topic_name

Name of the **Topic** (p. 1258) related to the Content-filtered **Topic** (p. 1258).

6.22.2.3 System::String ^ 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).

6.22.2.4 System::String ^ DDS::ContentFilterProperty_t::filter_expression

The actual filter expression. Must be a valid expression for the filter class specified using filterClassName.

6.22.2.5 StringSeq ^ DDS::ContentFilterProperty_t::expression_parameters

Defines the value for each parameter in the filter expression.

6.23 DDS::DatabaseQosPolicy Class Reference

Various threads and resource limits settings used by RTI Data Distribution Service to control its internal database.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_database_qos_policy_name** ()

  *Stringified human-readable name for **DDS::DatabaseQosPolicy*** (p. 428).

Public Attributes

- **ThreadSettings_t^ thread**

  Database thread settings.

- **Duration_t shutdown_timeout**

  The maximum wait time during a shutdown.

- **Duration_t cleanup_period**

  The database thread will wake up at this rate to clean up the database.

- **Duration_t shutdown_cleanup_period**

  The clean-up period used during database shut-down.

- System::Int32 **initial_records**

  The initial number of total records.

- System::Int32 **max_skiplist_level**

  The maximum level of the skiplist.

- System::Int32 **max_weak_references**

  The maximum number of weak references.

- System::Int32 **initial_weak_references**

  The initial number of weak references.

6.23.1 Detailed Description

Various threads and resource limits settings used by RTI Data Distribution Service 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. 577) that maintains this database is deleted, it shuts down this thread.

The Database QoS policy is used to configure how RTI Data Distribution Service 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. 430) and **DDS::DatabaseQosPolicy::shutdown_cleanup_period** (p. 430) parameters to decrease the time it takes to delete a **DDS::DomainParticipant** (p. 577) when your application is shutting down.

The **DDS::DomainParticipantResourceLimitsQosPolicy** (p. 688) controls the memory allocation for elements stored in the database.

This QoS policy is an extension to the DDS standard.

Entity:

DDS::DomainParticipant (p. 577)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) **NO** (p. 269)

6.23.2 Member Data Documentation

6.23.2.1 ThreadSettings_t ^ DDS::DatabaseQosPolicy::thread

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 Solaris: OS default priority

For Linux: OS default priority

For LynxOS: 10

For INTEGRITY: 60

For VxWorks: 120

For all others: OS default priority.

[default] The actual value depends on your architecture:

For Windows: OS default stack size

For Solaris: OS default stack size

For Linux: OS default stack size

For LynxOS: 16∗1024

For INTEGRITY: 20∗1024

For VxWorks: 16∗1024

For all others: OS default stack size.

[default] mask DDS::ThreadSettingsKind::THREAD_SETTINGS_STDIO

6.23.2.2 Duration_t DDS::DatabaseQosPolicy::shutdown_timeout

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_t::DURATION_INFINITE** (p. 253)]

6.23.2.3 Duration_t DDS::DatabaseQosPolicy::cleanup_period

The database thread will wake up at this rate to clean up the database.

[default] 61 seconds

[range] [0,1 year]

6.23.2.4 Duration_t DDS::DatabaseQosPolicy::shutdown_cleanup_period

The clean-up period used during database shut-down.

[default] 1 second

[**range**] [0,1 year]

6.23.2.5 System::Int32 DDS::DatabaseQosPolicy::initial_records

The initial number of total records.

[**default**] 1024

[**range**] [1,10 million]

6.23.2.6 System::Int32 DDS::DatabaseQosPolicy::max_skiplist_level

The maximum level of the skiplist.

The skiplist is used to keep records in the database. Usually, the search time is log2(N), where N is the total number of records in one skiplist. However, once N exceeds 2^n, where n is the maximum skiplist level, the search time will become more and more linear. Therefore, the maximum level should be set such that 2^n is larger than the maximum(N among all skiplists). Usually, the maximum N is the maximum number of remote and local writers or readers.

[**default**] 14

[**range**] [1,31]

6.23.2.7 System::Int32 DDS::DatabaseQosPolicy::max_weak_references

The maximum number of weak references.

A weak reference is an internal data structure that refers to a record within RTI Data Distribution Service' internal database. This field configures the maximum number of such references that RTI Data Distribution Service may create.

The actual number of weak references is permitted to grow from an initial value (indicated by **DDS::DatabaseQosPolicy::initial_weak_references** (p. 432)) to this maximum. To prevent RTI Data Distribution Service 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. 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 Data Distribution Service' 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

[range] [1, 100 million] or DDS::LENGTH_UNLIMITED, >= initial_weak_references

See also:

DDS::DatabaseQosPolicy::initial_weak_references (p. 432)

6.23.2.8 System::Int32 DDS::DatabaseQosPolicy::initial_weak_references

The initial number of weak references.

See **DDS::DatabaseQosPolicy::max_weak_references** (p. 431) 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. 431) using RTI Data Distribution Service' 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 REDA 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. 431)

6.24 DDS::DataReader Class Reference

<<***interface***>> *(p. 175)* 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. 1201).

`#include <managed_subscription.h>`

Inheritance diagram for DDS::DataReader::

Public Member Functions

- ^ **ReadCondition**^ **create_readcondition** (System::UInt32 sample_states, System::UInt32 view_states, System::UInt32 instance_states)

  *Creates a **DDS::ReadCondition** (p. 1084).*

- ^ **QueryCondition**^ **create_querycondition** (System::UInt32 sample_states, System::UInt32 view_states, System::UInt32 instance_states, System::String^ query_expression, **StringSeq**^ query_parameters)

  *Creates a **DDS::QueryCondition** (p. 1082).*

- ^ void **delete_readcondition** (**ReadCondition**^ %condition)

  *Deletes a **DDS::ReadCondition** (p. 1084) or **DDS::QueryCondition** (p. 1082) attached to the **DDS::DataReader** (p. 433).*

- ^ void **delete_contained_entities** ()

  *Deletes all the entities that were created by means of the "create" operations on the **DDS::DataReader** (p. 433).*

- ^ void **wait_for_historical_data** (**Duration_t**% max_wait)

  *Waits until all "historical" data is received for **DDS::DataReader** (p. 433) entities that have a non-VOLATILE PERSISTENCE Qos kind.*

- ^ void **get_matched_publications** (**InstanceHandleSeq**^ publication_handles)

  *Retrieve the list of publications currently "associated" with this **DDS::DataReader** (p. 433).*

- ^ void **get_matched_publication_data** (**PublicationBuiltinTopicData**^ publication_data, **InstanceHandle_t**% publication_handle)

  *This operation retrieves the information on a publication that is currently "associated" with the **DDS::DataReader** (p. 433).*

^ **ITopicDescription**^ **get_topicdescription** ()

Returns the DDS::TopicDescription associated with the ***DDS::DataReader*** *(*p. 433*).*

^ **Subscriber**^ **get_subscriber** ()

Returns the ***DDS::Subscriber*** *(*p. 1201*) to which the* ***DDS::DataReader*** *(*p. 433*) belongs.*

^ void **get_sample_rejected_status** (**SampleRejectedStatus**% status)

Accesses the DDS::StatusKind::SAMPLE_REJECTED_STATUS communication status.

^ void **get_liveliness_changed_status** (**LivelinessChangedStatus**% status)

Accesses the DDS::StatusKind::LIVELINESS_CHANGED_STATUS communication status.

^ void **get_requested_deadline_missed_status** (**RequestedDeadlineMissedStatus**% status)

Accesses the DDS::StatusKind::REQUESTED_DEADLINE_MISSED_STATUS communication status.

^ void **get_requested_incompatible_qos_status** (**RequestedIncompatibleQosStatus**^ status)

Accesses the DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS communication status.

^ void **get_sample_lost_status** (**SampleLostStatus**% status)

Accesses the DDS::StatusKind::SAMPLE_LOST_STATUS communication status.

^ void **get_subscription_matched_status** (**SubscriptionMatchedStatus**% status)

Accesses the DDS::StatusKind::SUBSCRIPTION_MATCHED_STATUS communication status.

^ virtual void **get_datareader_cache_status** (**DataReaderCacheStatus**% status)

<<**eXtension**>> (p. 174) *Get the datareader cache status for this reader.*

^ virtual void **get_datareader_protocol_status** (**DataReaderProtocolStatus**% status)

<<**eXtension**>> (p. 174) *Get the datareader protocol status for this reader.*

^ virtual void **get_matched_publication_datareader_protocol_status** (**DataReaderProtocolStatus**% status, **InstanceHandle_t**% publication_handle)

<<**eXtension**>> (p. 174) *Get the datareader protocol status for this reader, per matched publication identified by the publication_handle.*

^ void **set_qos** (**DataReaderQos**^ qos)

Sets the reader QoS.

^ void **set_qos_with_profile** (System::String^ library_name, System::String^ profile_name)

<<**eXtension**>> (p. 174) *Change the QoS of this reader using the input XML QoS profile.*

^ void **get_qos** (**DataReaderQos**^ qos)

Gets the reader QoS.

^ void **set_listener** (**DataReaderListener**^ l, **StatusMask** mask)

Sets the reader listener.

^ **DataReaderListener**^ **get_listener** ()

Get the reader listener.

^ template<typename T>
void **read_untyped** (**DDS::LoanableSequence**< T >^received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, System::UInt32 sample_states, System::UInt32 view_states, System::UInt32 instance_states)

Read data samples, if any are available.

^ template<typename T>
void **take_untyped** (**DDS::LoanableSequence**< T >^received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, System::UInt32 sample_states, System::UInt32 view_states, System::UInt32 instance_states)

Take data samples, if any are available.

^ template<typename T>
void **read_w_condition_untyped** (**DDS::LoanableSequence**< T >^received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, **DDS::ReadCondition**^ condition)

Read data samples, if any are available.

^ template<typename T>
void **take_w_condition_untyped** (**DDS::LoanableSequence**< T >^received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, **DDS::ReadCondition**^ condition)

Take data samples, if any are available.

^ void **read_next_sample_untyped** (System::Object^ received_data, **DDS::SampleInfo**^ sample_info)

Read data samples, if any are available.

^ void **take_next_sample_untyped** (System::Object^ received_data, **DDS::SampleInfo**^ sample_info)

Take data samples, if any are available.

^ template<typename T>
void **read_instance_untyped** (**DDS::LoanableSequence**< T >^received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, **DDS::InstanceHandle_t**% a_handle, System::UInt32 sample_states, System::UInt32 view_states, System::UInt32 instance_states)

Read data samples, if any are available.

^ template<typename T>
void **take_instance_untyped** (**DDS::LoanableSequence**< T >^received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, **DDS::InstanceHandle_t**% a_handle, System::UInt32 sample_states, System::UInt32 view_states, System::UInt32 instance_states)

Take data samples, if any are available.

^ template<typename T>
void **read_next_instance_untyped** (**DDS::LoanableSequence**< T >^received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, **DDS::InstanceHandle_t**% previous_handle, System::UInt32 sample_states, System::UInt32 view_states, System::UInt32 instance_states)

Read data samples, if any are available.

^ template<typename T>
void **take_next_instance_untyped** (**DDS::LoanableSequence**< T >^received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, **DDS::InstanceHandle_t**% previous_handle, System::UInt32 sample_states, System::UInt32 view_states, System::UInt32 instance_states)

Take data samples, if any are available.

^ template<typename T>
void **read_next_instance_w_condition_untyped** (**DDS::LoanableSequence**< T >^received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, **DDS::InstanceHandle_t**% previous_handle, **DDS::ReadCondition**^ condition)

Read data samples, if any are available.

^ template<typename T>
void **take_next_instance_w_condition_untyped** (**DDS::LoanableSequence**< T >^received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, **DDS::InstanceHandle_t**% previous_handle, **DDS::ReadCondition**^ condition)

Take data samples, if any are available.

^ template<typename T>
void **return_loan_untyped** (**DDS::LoanableSequence**< T >^received_data, **DDS::SampleInfoSeq**^ info_seq)

Return loaned sample data and meta-data.

^ void **get_key_value_untyped** (System::Object^ key_holder, **DDS::InstanceHandle_t**% handle)

Fill in the key fields of the given data sample.

^ virtual void **enable** () override

*Enables the **DDS::Entity** (p. 845).*

^ virtual **StatusCondition**^ **get_statuscondition** () override

*Allows access to the **DDS::StatusCondition** (p. 1183) associated with the **DDS::Entity** (p. 845).*

^ virtual **StatusMask get_status_changes** () override

*Retrieves the list of communication statuses in the **DDS::Entity** (p. 845) that are triggered.*

^ virtual **InstanceHandle_t get_instance_handle** () override

*Allows access to the **DDS::InstanceHandle_t** (p. 905) associated with the **DDS::Entity** (p. 845).*

6.24.1 Detailed Description

<<interface>> *(p. 175)* 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. 1201).

QoS:

DDS::DataReaderQos (p. 480)

Status:

DDS::StatusKind::DATA_AVAILABLE_STATUS;
DDS::StatusKind::LIVELINESS_CHANGED_STATUS, **DDS::LivelinessChangedStatus** (p. 956);
DDS::StatusKind::REQUESTED_DEADLINE_MISSED_STATUS, **DDS::RequestedDeadlineMissedStatus** (p. 1105);
DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS, **DDS::RequestedIncompatibleQosStatus** (p. 1107);
DDS::StatusKind::SAMPLE_LOST_STATUS, **DDS::SampleLostStatus** (p. 1158);
DDS::StatusKind::SAMPLE_REJECTED_STATUS, **DDS::SampleRejectedStatus** (p. 1159);
DDS::StatusKind::SUBSCRIPTION_MATCHED_STATUS, DDS::SubscriptionMatchedStatus

Listener:

DDS::DataReaderListener (p. 461)

A **DDS::DataReader** (p. 433) refers to exactly one DDS::TopicDescription (either a **DDS::Topic** (p. 1258), a **DDS::ContentFilteredTopic** (p. 419) or a **DDS::MultiTopic** (p. 984)) 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. 433) is an abstract class. It must be specialised for each particular application data-type (see **USER_DATA** (p. 273)). The additional methods or functions that must be defined in the auto-generated class for a hypothetical application type `Foo` (p. 877) are specified in the generic type **DDS::TypedDataReader** (p. 1338).

The following operations may be called even if the **DDS::DataReader** (p. 433) is not enabled. Other operations will fail with the value **DDS::Retcode_NotEnabled** (p. 1121) if called on a disabled **DDS::DataReader** (p. 433):

- The base-class operations **DDS::DataReader::set_qos** (p. 448), **DDS::DataReader::get_qos** (p. 449), **DDS::DataReader::set_listener** (p. 450), **DDS::DataReader::get_listener** (p. 450), **DDS::Entity::enable** (p. 848), **DDS::Entity::get_statuscondition** (p. 849) and **DDS::Entity::get_status_changes** (p. 850)
- **DDS::DataReader::get_liveliness_changed_status** (p. 445) **DDS::DataReader::get_requested_deadline_missed_status** (p. 445) **DDS::DataReader::get_requested_incompatible_qos_status** (p. 445) **DDS::DataReader::get_sample_lost_status** (p. 446) **DDS::DataReader::get_sample_rejected_status** (p. 444) **DDS::DataReader::get_subscription_matched_status** (p. 446)

All sample-accessing operations, namely: **DDS::TypedDataReader::read** (p. 1341), **DDS::TypedDataReader::take** (p. 1342), **DDS::TypedDataReader::read_w_condition** (p. 1349), and **DDS::TypedDataReader::take_w_condition** (p. 1350) may fail with the error **DDS::Retcode_PreconditionNotMet** (p. 1123) as described in **DDS::Subscriber::begin_access** (p. 1216).

See also:

Operations Allowed in Listener Callbacks (p. 954)

Examples:

HelloWorld_subscriber.cpp, and **HelloWorldSupport.cpp**.

6.24.2 Member Function Documentation

6.24.2.1 ReadCondition ^ DDS::DataReader::create_readcondition (System::UInt32 *sample_states*, System::UInt32 *view_states*, System::UInt32 *instance_states*)

Creates a **DDS::ReadCondition** (p. 1084).

The returned **DDS::ReadCondition** (p. 1084) will be attached and belong to the **DDS::DataReader** (p. 433).

Parameters:

sample_states *<<in>>* *(p. 175)* sample state of the data samples that are of interest

view_states *<<in>>* *(p. 175)* view state of the data samples that are of interest

instance_states *<<in>>* *(p. 175)* instance state of the data samples that are of interest

Returns:

return **DDS::ReadCondition** (p. 1084) created. Returns NULL in case of failure.

6.24.2.2 QueryCondition ^ DDS::DataReader::create_querycondition (System::UInt32 *sample_states*, System::UInt32 *view_states*, System::UInt32 *instance_states*, System::String^ *query_expression*, StringSeq^ *query_parameters*)

Creates a **DDS::QueryCondition** (p. 1082).

The returned **DDS::QueryCondition** (p. 1082) will be attached and belong to the **DDS::DataReader** (p. 433).

Queries and Filters Syntax (p. 184) describes the syntax of `query_expression` and `query_parameters`.

Parameters:

sample_states *<<in>> (p. 175)* sample state of the data samples that are of interest

view_states *<<in>> (p. 175)* view state of the data samples that are of interest

instance_states *<<in>> (p. 175)* instance state of the data samples that are of interest

query_expression *<<in>> (p. 175)* Expression for the query. Cannot be NULL.

query_parameters *<<in>> (p. 175)* Parameters for the query expression. Cannot be NULL.

Returns:

NULL

6.24.2.3 void DDS::DataReader::delete_readcondition (ReadCondition^ % *condition*)

Deletes a **DDS::ReadCondition** (p. 1084) or **DDS::QueryCondition** (p. 1082) attached to the **DDS::DataReader** (p. 433).

Since **DDS::QueryCondition** (p. 1082) specializes **DDS::ReadCondition** (p. 1084), it can also be used to delete a **DDS::QueryCondition** (p. 1082).

Precondition:

The **DDS::ReadCondition** (p. 1084) must be attached to the **DDS::DataReader** (p. 433), or the operation will fail with the error **DDS::Retcode_PreconditionNotMet** (p. 1123).

Parameters:

condition *<<in>>* *(p. 175)* **Condition** (p. 408) to be deleted.

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_PreconditionNotMet** (p. 1123)

6.24.2.4 void DDS::DataReader::delete_contained_entities ()

Deletes all the entities that were created by means of the "create" operations on the **DDS::DataReader** (p. 433).

Deletes all contained **DDS::ReadCondition** (p. 1084) and **DDS::QueryCondition** (p. 1082) objects.

The operation will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123) if the any of the contained entities is in a state where it cannot be deleted.

Once **DDS::DataReader::delete_contained_entities** (p. 441) completes successfully, the application may delete the **DDS::DataReader** (p. 433), knowing that it has no contained **DDS::ReadCondition** (p. 1084) and **DDS::QueryCondition** (p. 1082) objects.

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_PreconditionNotMet** (p. 1123)

6.24.2.5 void DDS::DataReader::wait_for_historical_data (Duration_t% *max_wait*)

Waits until all "historical" data is received for **DDS::DataReader** (p. 433) entities that have a non-VOLATILE PERSISTENCE Qos kind.

This operation is intended only for **DDS::DataReader** (p. 433) entities that have a non-VOLATILE PERSISTENCE QoS kind.

As soon as an application enables a non-VOLATILE **DDS::DataReader** (p. 433), it will start receiving both "historical" data, i.e. the data that was

written prior to the time the **DDS::DataReader** (p. 433) joined the domain, as well as any new data written by the **DDS::DataWriter** (p. 499) 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. 441) operations.

The operation **DDS::DataReader::wait_for_historical_data** (p. 441) blocks the calling thread until either all "historical" data is received, or else duration specified by the max_wait parameter clapses, whichever happens first. A successful completion indicates that all the "historical" data was "received"; timing out indicates that max_wait elapsed before all the data was received.

Parameters:

max_wait *<<in>> (p. 175)* Timeout value. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_Timeout** (p. 1124) or **DDS::Retcode_NotEnabled** (p. 1121).

6.24.2.6 void DDS::DataReader::get_matched_publications (InstanceHandleSeq^ *publication_handles*)

Retrieve the list of publications currently "associated" with this **DDS::DataReader** (p. 433).

Matching publications are those in the same domain that have a matching **DDS::Topic** (p. 1258), compatible QoS common partition that the **DDS::DomainParticipant** (p. 577) has not indicated should be "ignored" by means of the **DDS::DomainParticipant::ignore_publication** (p. 635) operation.

The handles returned in the `publication_handles'` list are the ones that are used by the DDS implementation to locally identify the corresponding matched **DDS::DataWriter** (p. 499) entities. These handles match the ones that appear in the `instance_handle` field of the **DDS::SampleInfo** (p. 1148) when reading the **DDS::PublicationBuiltinTopicDataTypeSupport::PUBLICATION_TOPIC_NAME** (p. 230) builtin topic

Parameters:

publication_handles **inout**.

The sequence will be grown if the sequence has ownership and the system has the corresponding resources. Use a sequence without ownership to avoid dynamic

memory allocation. If the sequence is too small to store all the matches and the system can not resize the sequence, this method will fail with **DDS::Retcode_OutOfResources** (p. 1122).

The maximum number of matches possible is configured with **DDS::DomainParticipantResourceLimitsQosPolicy** (p. 688). You can use a zero-maximum sequence without ownership to quickly check whether there are any matches without allocating any memory. Cannot be NULL..

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_OutOfResources** (p. 1122) if the sequence is too small and the system can not resize it, or **DDS::Retcode_NotEnabled** (p. 1121)

6.24.2.7 void DDS::DataReader::get_matched_publication_data (PublicationBuiltinTopicData^ *publication_data*, InstanceHandle_t% *publication_handle*)

This operation retrieves the information on a publication that is currently "associated" with the **DDS::DataReader** (p. 433).

Publication with a matching **DDS::Topic** (p. 1258), compatible QoS and common partition that the application has not indicated should be "ignored" by means of the **DDS::DomainParticipant::ignore_publication** (p. 635) operation.

The `publication_handle` must correspond to a publication currently associated with the **DDS::DataReader** (p. 433). Otherwise, the operation will fail with **DDS::Retcode_BadParameter** (p. 1115). Use the operation **DDS::DataReader::get_matched_publications** (p. 442) to find the publications that are currently matched with the **DDS::DataReader** (p. 433).

Note: This operation does not retrieve the following information in **DDS::PublicationBuiltinTopicData** (p. 1030):

- **DDS::PublicationBuiltinTopicData::type_code** (p. 1036)
- DDS::PublicationBuiltinTopicData::property

The above information is available through **DDS::DataReaderListener::on_data_available()** (p. 463) (if a reader listener is installed on the **DDS::PublicationBuiltinTopicDataDataReader** (p. 1038)).

Parameters:

publication_data *<<inout>>* *(p. 176)*. The information to be filled in on the associated publication. Cannot be NULL.

publication_handle *<<in>>* *(p. 175)*. Handle to a specific publication associated with the **DDS::DataWriter** (p. 499). Cannot be NULL.. Must correspond to a publication currently associated with the **DDS::DataReader** (p. 433).

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_NotEnabled** (p. 1121)

6.24.2.8 ITopicDescription ^ DDS::DataReader::get_topicdescription ()

Returns the DDS::TopicDescription associated with the **DDS::DataReader** (p. 433).

Returns that same DDS::TopicDescription that was used to create the **DDS::DataReader** (p. 433).

Returns:

DDS::TopicDescription associated with the **DDS::DataReader** (p. 433).

6.24.2.9 Subscriber ^ DDS::DataReader::get_subscriber ()

Returns the **DDS::Subscriber** (p. 1201) to which the **DDS::DataReader** (p. 433) belongs.

Returns:

DDS::Subscriber (p. 1201) to which the **DDS::DataReader** (p. 433) belongs.

6.24.2.10 void DDS::DataReader::get_sample_rejected_status (SampleRejectedStatus% *status*)

Accesses the DDS::StatusKind::SAMPLE_REJECTED_STATUS communication status.

Parameters:

status *<<inout>>* *(p. 176)* **DDS::SampleRejectedStatus** (p. 1159) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.24.2.11 void DDS::DataReader::get_liveliness_changed_status (LivelinessChangedStatus% *status*)

Accesses the DDS::StatusKind::LIVELINESS_CHANGED_STATUS communication status.

Parameters:

status *<<inout>> (p. 176)* **DDS::LivelinessChangedStatus** (p. 956) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.24.2.12 void DDS::DataReader::get_requested_deadline_missed_status (RequestedDeadlineMissedStatus% *status*)

Accesses the DDS::StatusKind::REQUESTED_DEADLINE_MISSED_STATUS communication status.

Parameters:

status *<<inout>> (p. 176)* **DDS::RequestedDeadlineMissedStatus** (p. 1105) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.24.2.13 void DDS::DataReader::get_requested_incompatible_qos_status (RequestedIncompatibleQosStatus^ *status*)

Accesses the DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS communication status.

Parameters:

status *<<inout>> (p. 176)* **DDS::RequestedIncompatibleQosStatus** (p. 1107) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.24.2.14 void DDS::DataReader::get_sample_lost_status (SampleLostStatus% *status*)

Accesses the DDS::StatusKind::SAMPLE_LOST_STATUS communication status.

Parameters:

status <<*inout*>> *(p. 176)* **DDS::SampleLostStatus** (p. 1158) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.24.2.15 void DDS::DataReader::get_subscription_matched_status (SubscriptionMatchedStatus% *status*)

Accesses the DDS::StatusKind::SUBSCRIPTION_MATCHED_STATUS communication status.

Parameters:

status <<*inout*>> *(p. 176)* **DDS::SubscriptionMatchedStatus** (p. 1244) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.24.2.16 virtual void DDS::DataReader::get_datareader_cache_status (DataReaderCacheStatus% *status*) [virtual]

<<*eXtension*>> *(p. 174)* Get the datareader cache status for this reader.

Parameters:

status <<*inout*>> *(p. 176)* **DDS::DataReaderCacheStatus** (p. 460) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-NotEnabled** (p. 1121).

6.24.2.17 virtual void DDS::DataReader::get_datareader_-protocol_status (DataReaderProtocolStatus% *status*)

`[virtual]`

*<<**eXtension**>> (p. 174)* Get the datareader protocol status for this reader.

Parameters:

status *<<**inout**>> (p. 176)* **DDS::DataReaderProtocolStatus** (p. 470) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-NotEnabled** (p. 1121).

6.24.2.18 virtual void DDS::DataReader::get_matched_publication_-datareader_protocol_status (DataReaderProtocolStatus% *status*, InstanceHandle_t% *publication_handle*)

`[virtual]`

*<<**eXtension**>> (p. 174)* Get the datareader protocol status for this reader, per matched publication identified by the publication_handle.

Note: Status for a remote entity is only kept while the entity is alive. Once a remote entity is no longer alive, its status is deleted.

Parameters:

status *<<**inout**>> (p. 176)*. The information to be filled in on the associated publication. Cannot be NULL.

publication_handle *<<**in**>> (p. 175)*. Handle to a specific publication associated with the **DDS::DataWriter** (p. 499). Cannot be NULL.. Must correspond to a publication currently associated with the **DDS::DataReader** (p. 433).

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-NotEnabled** (p. 1121)

6.24.2.19 void DDS::DataReader::set_qos (DataReaderQos^ *qos*)

Sets the reader QoS.

This operation modifies the QoS of the **DDS::DataReader** (p. 433).

The **DDS::DataReaderQos::user_data** (p. 483), **DDS::DataReaderQos::deadline** (p. 482), **DDS::DataReaderQos::latency_budget** (p. 482), **DDS::DataReaderQos::time_based_filter** (p. 483), **DDS::DataReaderQos::reader_data_lifecycle** (p. 483) can be changed. The other policies are immutable.

Parameters:

qos *<<in>> (p. 175)* The **DDS::DataReaderQos** (p. 480) to be set to. Policies must be consistent. Immutable policies cannot be changed after **DDS::DataReader** (p. 433) is enabled. The special value **DDS::Subscriber::DATAREADER_QOS_DEFAULT** (p. 95) can be used to indicate that the QoS of the **DDS::DataReader** (p. 433) should be changed to match the current default **DDS::DataReaderQos** (p. 480) set in the **DDS::Subscriber** (p. 1201). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_ImmutablePolicy** (p. 1118), or **DDS::Retcode_InconsistentPolicy** (p. 1119).

See also:

DDS::DataReaderQos (p. 480) for rules on consistency among QoS
set_qos (abstract) (p. 846)
DDS::DataReader::set_qos (p. 448)
Operations Allowed in Listener Callbacks (p. 954)

6.24.2.20 void DDS::DataReader::set_qos_with_profile (System::String^ *library_name*, System::String^ *profile_name*)

<<eXtension>> (p. 174) Change the QoS of this reader using the input XML QoS profile.

This operation modifies the QoS of the **DDS::DataReader** (p. 433).

The **DDS::DataReaderQos::user_data** (p. 483), **DDS::DataReaderQos::deadline** (p. 482), **DDS::DataReaderQos::latency_budget** (p. 482), **DDS::DataReaderQos::time_based_filter** (p. 483),

DDS::DataReaderQos::reader_data_lifecycle (p. 483) can be changed. The other policies are immutable.

Parameters:

library_name *<<in>>* *(p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::Subscriber::set_default_library** (p. 1208)).

profile_name *<<in>>* *(p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::Subscriber::set_default_profile** (p. 1209)).

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_ImmutablePolicy** (p. 1118), or **DDS::Retcode_InconsistentPolicy** (p. 1119).

See also:

DDS::DataReaderQos (p. 480) for rules on consistency among QoS
DDS::DataReader::set_qos (p. 448)
Operations Allowed in Listener Callbacks (p. 954)

6.24.2.21 void DDS::DataReader::get_qos (DataReaderQos^ *qos*)

Gets the reader QoS.

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

Parameters:

qos *<<inout>>* *(p. 176)* The **DDS::DataReaderQos** (p. 480) to be filled up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

get_qos (abstract) (p. 847)

6.24.2.22 void DDS::DataReader::set_listener (DataReaderListener^ *l*, StatusMask *mask*)

Sets the reader listener.

Parameters:

l <<*in*>> *(p. 175)* **DDS::DataReaderListener** (p. 461) to set to

mask <<*in*>> *(p. 175)* DDS::StatusMask associated with the **DDS::DataReaderListener** (p. 461).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

set_listener (abstract) (p. 847)

6.24.2.23 DataReaderListener ^ DDS::DataReader::get_listener ()

Get the reader listener.

Returns:

DDS::DataReaderListener (p. 461) of the **DDS::DataReader** (p. 433).

See also:

get_listener (abstract) (p. 848)

6.24.2.24 template<typename T> void DDS::DataReader::read_untyped (DDS::LoanableSequence< T >^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, System::UInt32 *sample_states*, System::UInt32 *view_states*, System::UInt32 *instance_states*) [explicit]

Read data samples, if any are available.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataReader** (p. 1338) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataReader::read** (p. 1341) method instead of this one. See that method for detailed documentation.

See also:

DDS::DataReader::take_untyped (p. 451)
DDS::TypedDataReader::read (p. 1341)

6.24.2.25 template<typename T> void DDS::DataReader::take_untyped (DDS::LoanableSequence< T >^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, System::UInt32 *sample_states*, System::UInt32 *view_states*, System::UInt32 *instance_states*) [explicit]

Take data samples, if any are available.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataReader** (p. 1338) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataReader::take** (p. 1342) method instead of this one. See that method for detailed documentation.

See also:

DDS::DataReader::read_untyped (p. 450)
DDS::TypedDataReader::take (p. 1342)

6.24.2.26 template<typename T> void DDS::DataReader::read_w_condition_untyped (DDS::LoanableSequence< T >^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, DDS::ReadCondition^ *condition*) [explicit]

Read data samples, if any are available.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataReader** (p. 1338) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataReader::read_w_condition** (p. 1349) method instead of this one. See that method for detailed documentation.

See also:

DDS::DataReader::take_w_condition_untyped (p. 452)
DDS::TypedDataReader::read_w_condition (p. 1349)

6.24.2.27 template<typename T> void DDS::DataReader::take_w_condition_untyped (DDS::LoanableSequence< T >^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, DDS::ReadCondition^ *condition*) [explicit]

Take data samples, if any are available.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataReader** (p. 1338) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataReader::take_w_condition** (p. 1350) method instead of this one. See that method for detailed documentation.

See also:

DDS::DataReader::read_w_condition_untyped (p. 451)
DDS::TypedDataReader::take_w_condition (p. 1350)

6.24.2.28 void DDS::DataReader::read_next_sample_untyped (System::Object^ *received_data*, DDS::SampleInfo^ *sample_info*)

Read data samples, if any are available.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataReader** (p. 1338) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataReader::read_next_sample** (p. 1351) method instead of this one. See that method for detailed documentation.

See also:

DDS::DataReader::take_next_sample_untyped (p. 452)
DDS::TypedDataReader::read_next_sample (p. 1351)

6.24.2.29 void DDS::DataReader::take_next_sample_untyped (System::Object^ *received_data*, DDS::SampleInfo^ *sample_info*)

Take data samples, if any are available.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataReader** (p. 1338) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataReader::take_next_sample** (p. 1352) method instead of this one. See that method for detailed documentation.

See also:

DDS::DataReader::read_next_sample_untyped (p. 452)
DDS::TypedDataReader::take_next_sample (p. 1352)

6.24.2.30 template<typename T> void DDS::DataReader::read_instance_untyped (DDS::LoanableSequence< T >^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, DDS::InstanceHandle_t% *a_handle*, System::UInt32 *sample_states*, System::UInt32 *view_states*, System::UInt32 *instance_states*) [explicit]

Read data samples, if any are available.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataReader** (p. 1338) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataReader::read_instance** (p. 1353) method instead of this one. See that method for detailed documentation.

See also:

DDS::DataReader::take_instance_untyped (p. 453)
DDS::TypedDataReader::read_instance (p. 1353)

6.24.2.31 template<typename T> void DDS::DataReader::take_instance_untyped (DDS::LoanableSequence< T >^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, DDS::InstanceHandle_t% *a_handle*, System::UInt32 *sample_states*, System::UInt32 *view_states*, System::UInt32 *instance_states*) [explicit]

Take data samples, if any are available.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataReader** (p. 1338) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataReader::take_instance** (p. 1355) method instead of this one. See that method for detailed documentation.

See also:

DDS::DataReader::read_instance_untyped (p. 453)
DDS::TypedDataReader::take_instance (p. 1355)

6.24.2.32 template<typename T> void DDS::DataReader::read_next_instance_untyped (DDS::LoanableSequence< T >^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, DDS::InstanceHandle_t% *previous_handle*, System::UInt32 *sample_states*, System::UInt32 *view_states*, System::UInt32 *instance_states*) [explicit]

Read data samples, if any are available.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataReader** (p. 1338) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataReader::read_next_instance** (p. 1357) method instead of this one. See that method for detailed documentation.

See also:

DDS::DataReader::take_next_instance_untyped (p. 454)
DDS::TypedDataReader::read_next_instance (p. 1357)

6.24.2.33 template<typename T> void DDS::DataReader::take_next_instance_untyped (DDS::LoanableSequence< T >^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, DDS::InstanceHandle_t% *previous_handle*, System::UInt32 *sample_states*, System::UInt32 *view_states*, System::UInt32 *instance_states*) [explicit]

Take data samples, if any are available.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataReader** (p. 1338) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataReader::take_next_instance** (p. 1359) method instead of this one. See that method for detailed documentation.

See also:

DDS::DataReader::read_next_instance_untyped (p. 454)
DDS::TypedDataReader::take_next_instance (p. 1359)

6.24.2.34 template<typename T> void DDS::DataReader::read_next_instance_w_condition_untyped (DDS::LoanableSequence< T >^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, DDS::InstanceHandle_t% *previous_handle*, DDS::ReadCondition^ *condition*) [explicit]

Read data samples, if any are available.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataReader** (p. 1338) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataReader::read_next_instance_w_condition** (p. 1361) method instead of this one. See that method for detailed documentation.

See also:

DDS::DataReader::take_next_instance_w_condition_untyped (p. 455)
DDS::TypedDataReader::read_next_instance_w_condition (p. 1361)

6.24.2.35 template<typename T> void DDS::DataReader::take_next_instance_w_condition_untyped (DDS::LoanableSequence< T >^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, DDS::InstanceHandle_t% *previous_handle*, DDS::ReadCondition^ *condition*) [explicit]

Take data samples, if any are available.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataReader** (p. 1338) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataReader::take_next_instance_w_condition** (p. 1362) method instead of this one. See that method for detailed documentation.

See also:

DDS::DataReader::read_next_instance_w_condition_untyped (p. 455)
DDS::TypedDataReader::take_next_instance_w_condition (p. 1362)

6.24.2.36 template<typename T> void DDS::DataReader::return_loan_untyped (DDS::LoanableSequence< T >^ *received_data*, DDS::SampleInfoSeq^ *info_seq*) [explicit]

Return loaned sample data and meta-data.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataReader** (p. 1338) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataReader::return_loan** (p. 1364) method instead of this one. See that method for detailed documentation.

See also:

DDS::TypedDataReader::return_loan (p. 1364)

6.24.2.37 void DDS::DataReader::get_key_value_untyped (System::Object^ *key_holder*, DDS::InstanceHandle_t% *handle*)

Fill in the key fields of the given data sample.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataReader** (p. 1338) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataReader::get_key_value** (p. 1365) method instead of this one. See that method for detailed documentation.

See also:

DDS::TypedDataReader::get_key_value (p. 1365)

6.24.2.38 virtual void DDS::DataReader::enable () [override, virtual]

Enables the **DDS::Entity** (p. 845).

This operation enables the **Entity** (p. 845). **Entity** (p. 845) objects can be created either enabled or disabled. This is controlled by the value of the **ENTITY_FACTORY** (p. 304) QoS policy on the corresponding factory for the **DDS::Entity** (p. 845).

By default, **ENTITY_FACTORY** (p. 304) is set so that it is not necessary to explicitly call **DDS::Entity::enable** (p. 848) on newly created entities.

The **DDS::Entity::enable** (p. 848) operation is idempotent. Calling enable on an already enabled **Entity** (p. 845) returns OK and has no effect.

If a **DDS::Entity** (p. 845) 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. 849)
- 'factory' operations
- **DDS::Entity::get_status_changes** (p. 850) 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_NotEnabled** (p. 1121).

It is legal to delete an **DDS::Entity** (p. 845) that has not been enabled by calling the proper operation on its factory.

Entities created from a factory that is disabled are created disabled, regardless of the setting of the **DDS::EntityFactoryQosPolicy** (p. 851).

Calling enable on an **Entity** (p. 845) whose factory is not enabled will fail and return **DDS::Retcode_PreconditionNotMet** (p. 1123).

If **DDS::EntityFactoryQosPolicy::autoenable_created_entities** (p. 852) is TRUE, the enable operation on a factory will automatically enable all entities created from that factory.

Listeners associated with an entity are not called until the entity is enabled.

Conditions associated with a disabled entity are "inactive," that is, they have a `trigger_value` == FALSE.

Exceptions:

One of the **Standard Return Codes** (p. 235), **Standard Return Codes** (p. 235) or **DDS::Retcode_PreconditionNotMet** (p. 1123).

Implements **DDS::Entity** (p. 848).

6.24.2.39 virtual StatusCondition ^ DDS::DataReader::get_statuscondition () [override, virtual]

Allows access to the **DDS::StatusCondition** (p. 1183) associated with the **DDS::Entity** (p. 845).

The returned condition can then be added to a **DDS::WaitSet** (p. 1411) so that the application can wait for specific status changes that affect the **DDS::Entity** (p. 845).

Returns:

the status condition associated with this entity.

Implements **DDS::Entity** (p. 849).

6.24.2.40 virtual StatusMask DDS::DataReader::get_status_changes () [override, virtual]

Retrieves the list of communication statuses in the **DDS::Entity** (p. 845) that are triggered.

That is, the list of statuses whose value has changed since the last time the application read the status using the get_*_status() method.

When the entity is first created or if the entity is not enabled, all communication statuses are in the "untriggered" state so the list returned by the get_status_changes operation will be empty.

The list of statuses returned by the get_status_changes operation refers to the status that are triggered on the **Entity** (p. 845) itself and does not include statuses that apply to contained entities.

Returns:

list of communication statuses in the **DDS::Entity** (p. 845) that are triggered.

See also:

Status Kinds (p. 238)

Implements **DDS::Entity** (p. 850).

6.24.2.41 virtual InstanceHandle_t DDS::DataReader::get_instance_handle () [override, virtual]

Allows access to the **DDS::InstanceHandle_t** (p. 905) associated with the **DDS::Entity** (p. 845).

This operation returns the **DDS::InstanceHandle_t** (p. 905) that represents the **DDS::Entity** (p. 845).

Returns:

the instance handle associated with this entity.

Implements **DDS::Entity** (p. 850).

6.25 DDS::DataReaderCacheStatus Struct Reference

*<<**eXtension**>> (p. 174)* The status of the reader's cache.

`#include <managed_subscription.h>`

Public Attributes

- System::Int64 **sample_count_peak**

  The highest number of samples in the reader's queue over the lifetime of the reader.

- System::Int64 **sample_count**

  The number of samples in the reader's queue.

6.25.1 Detailed Description

*<<**eXtension**>> (p. 174)* The status of the reader's cache.

Entity:

DDS::DataReader (p. 433)

6.25.2 Member Data Documentation

6.25.2.1 System::Int64 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 System::Int64 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. 279)

6.26 DDS::DataReaderListener Class Reference

<<*interface*>> *(p. 175)* **DDS::Listener** (p. 952) for reader status.

`#include <managed_subscription.h>`

Inheritance diagram for DDS::DataReaderListener::

Public Member Functions

- ^ virtual void **on_requested_deadline_missed** (**DataReader**^ reader, **RequestedDeadlineMissedStatus**% status)

  Handles the DDS::StatusKind::REQUESTED_DEADLINE_MISSED_STATUS communication status.

- ^ virtual void **on_liveliness_changed** (**DataReader**^ reader, **LivelinessChangedStatus**% status)

  Handles the DDS::StatusKind::LIVELINESS_CHANGED_STATUS communication status.

- ^ virtual void **on_requested_incompatible_qos** (**DataReader**^ reader, **RequestedIncompatibleQosStatus**^ status)

  Handles the DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS communication status.

- ^ virtual void **on_sample_rejected** (**DataReader**^ reader, **SampleRejectedStatus**% status)

  Handles the DDS::StatusKind::SAMPLE_REJECTED_STATUS communication status.

- ^ virtual void **on_data_available** (**DataReader**^ reader)

  Handle the DDS::StatusKind::DATA_AVAILABLE_STATUS communication status.

- ^ virtual void **on_sample_lost** (**DataReader**^ reader, **SampleLostStatus**% status)

  Handles the DDS::StatusKind::SAMPLE_LOST_STATUS communication status.

- ^ virtual void **on_subscription_matched** (**DataReader**^ reader, **SubscriptionMatchedStatus**% status)

  Handles the DDS::StatusKind::SUBSCRIPTION_MATCHED_STATUS communication status.

6.26.1 Detailed Description

<<*interface*>> *(p. 175)* **DDS::Listener** (p. 952) for reader status.

Entity:

DDS::DataReader (p. 433)

Status:

DDS::StatusKind::DATA_AVAILABLE_STATUS;
DDS::StatusKind::LIVELINESS_CHANGED_STATUS,
DDS::LivelinessChangedStatus (p. 956);
DDS::StatusKind::REQUESTED_DEADLINE_MISSED_STATUS,
DDS::RequestedDeadlineMissedStatus (p. 1105);
DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS,
DDS::RequestedIncompatibleQosStatus (p. 1107);
DDS::StatusKind::SAMPLE_LOST_STATUS, **DDS::SampleLostStatus** (p. 1158);
DDS::StatusKind::SAMPLE_REJECTED_STATUS,
DDS::SampleRejectedStatus (p. 1159);
DDS::StatusKind::SUBSCRIPTION_MATCHED_STATUS,
DDS::SubscriptionMatchedStatus (p. 1244);

See also:

Status Kinds (p. 238)
Operations Allowed in Listener Callbacks (p. 954)

6.26.2 Member Function Documentation

6.26.2.1 virtual void DDS::DataReaderListener::on_requested_deadline_missed (DataReader^ *reader*, RequestedDeadlineMissedStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::REQUESTED_DEADLINE_MISSED_STATUS communication status.

Reimplemented in **DDS::DomainParticipantListener** (p. 681), and **DDS::SubscriberListener** (p. 1228).

6.26.2.2 virtual void DDS::DataReaderListener::on_liveliness_changed (DataReader^ *reader*, LivelinessChangedStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::LIVELINESS_CHANGED_STATUS communication status.

Reimplemented in **DDS::DomainParticipantListener** (p. 681), and **DDS::SubscriberListener** (p. 1228).

6.26.2.3 virtual void DDS::DataReaderListener::on_requested_incompatible_qos (DataReader^ *reader*, RequestedIncompatibleQosStatus^ *status*) [inline, virtual]

Handles the DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS communication status.

Reimplemented in **DDS::DomainParticipantListener** (p. 681), and **DDS::SubscriberListener** (p. 1228).

6.26.2.4 virtual void DDS::DataReaderListener::on_sample_rejected (DataReader^ *reader*, SampleRejectedStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::SAMPLE_REJECTED_STATUS communication status.

Reimplemented in **DDS::DomainParticipantListener** (p. 681), and **DDS::SubscriberListener** (p. 1228).

6.26.2.5 virtual void DDS::DataReaderListener::on_data_available (DataReader^ *reader*) [inline, virtual]

Handle the DDS::StatusKind::DATA_AVAILABLE_STATUS communication status.

Reimplemented in **DDS::DomainParticipantListener** (p. 682), and **DDS::SubscriberListener** (p. 1229).

6.26.2.6 virtual void DDS::DataReaderListener::on_sample_lost (DataReader^ *reader*, SampleLostStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::SAMPLE_LOST_STATUS communication status.

Reimplemented in **DDS::DomainParticipantListener** (p. 682), and **DDS::SubscriberListener** (p. 1229).

6.26.2.7 virtual void DDS::DataReaderListener::on_subscription_matched (DataReader^ *reader*, SubscriptionMatchedStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::SUBSCRIPTION_MATCHED_STATUS communication status.

Reimplemented in **DDS::DomainParticipantListener** (p. 682), and **DDS::SubscriberListener** (p. 1229).

6.27 DDS::DataReaderProtocolQosPolicy Struct Reference

Along with **DDS::WireProtocolQosPolicy** (p. 1423) and **DDS::DataWriterProtocolQosPolicy** (p. 529), this QoS policy configures the DDS on-the-network protocol (RTPS).

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_datareaderprotocol_qos_policy_name** ()

  *Stringified human-readable name for **DDS::DataReaderProtocolQosPolicy** (p. 465).*

Public Attributes

- **GUID_t virtual_guid**

  The virtual GUID (Global Unique Identifier).

- System::UInt32 **rtps_object_id**

  The RTPS Object ID.

- **RtpsReliableReaderProtocol_t rtps_reliable_reader**

  The reliable protocol defined in RTPS.

Properties

- System::Boolean **expects_inline_qos** `[get, set]`

  *Specifies whether this **DataReader** (p. 433) expects inline QoS with every sample.*

- System::Boolean **disable_positive_acks** `[get, set]`

  Whether the reader sends positive acknowledgements to writers.

- System::Boolean **propagate_dispose_of_unregistered_instances** `[get, set]`

  *Indicates whether or not an instance can move to the **DDS::InstanceStateKind::NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 909) state without being in the*

DDS::InstanceStateKind::ALIVE_INSTANCE_STATE *(p. 908)*
state.

6.27.1 Detailed Description

Along with **DDS::WireProtocolQosPolicy** (p. 1423) and **DDS::DataWriterProtocolQosPolicy** (p. 529), 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. 465) 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** (p. 499) or **DataReader** (p. 433) 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. 499) and **DDS::DataReader** (p. 433) 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 Data Distribution Service responds to "slow" reliable DataReaders or ones that disconnect or are otherwise lost. See **DDS::ReliabilityQosPolicy** (p. 1094) for more information on the per-DataReader/DataWriter reliability configuration. **DDS::HistoryQosPolicy** (p. 898) and **DDS::ResourceLimitsQosPolicy** (p. 1109) also play an important role in the DDS reliable protocol.

This QoS policy is an extension to the DDS standard.

Entity:

DDS::DataReader (p. 433)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)

6.27.2 Member Data Documentation

6.27.2.1 GUID_t DDS::DataReaderProtocolQosPolicy::virtual_guid

The virtual GUID (Global Unique Identifier).

The virtual GUID is used to uniquely identify different incarnations of the same **DDS::DataReader** (p. 433).

The association between a **DDS::DataReader** (p. 433) and its persisted state is done using the virtual GUID.

[**default**] DDS::Guid_t::GUID_AUTO

6.27.2.2 System::UInt32 DDS::DataReaderProtocolQosPolicy::rtps_object_id

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 Data Distribution Service 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 Data Distribution Service. In those cases, the recomendation is not to use automatic object ID assigment.

[**default**] **DDS::WireProtocolQosPolicy::RTPS_AUTO_ID** (p. 330)

[**range**] [0,0x00ffffff]

6.27.2.3 RtpsReliableReaderProtocol_t DDS::DataReaderProtocolQosPolicy::rtps_reliable_reader

The reliable protocol defined in RTPS.

[**default**] min_heartbeat_response_delay 0 seconds; max_heartbeat_response_delay 0.5 seconds

6.27.3 Property Documentation

6.27.3.1 System:: Boolean DDS::DataReaderProtocolQosPolicy::expects_inline_qos [get, set]

Specifies whether this **DataReader** (p. 433) expects inline QoS with every sample.

In RTI Data Distribution Service, a **DDS::DataReader** (p. 433) nominally relies on Discovery to propagate QoS on a matched **DDS::DataWriter** (p. 499). Alternatively, a **DDS::DataReader** (p. 433) may get information on a matched **DDS::DataWriter** (p. 499) through QoS sent inline with a sample.

Asserting **DDS::DataReaderProtocolQosPolicy::expects_inline_qos** (p. 468) indicates to a matching **DDS::DataWriter** (p. 499) that this **DDS::DataReader** (p. 433) expects to receive inline QoS with every sample. The complete set of inline QoS that a **DDS::DataWriter** (p. 499) may send inline is specified by the Real-Time Publish-Subscribe (RTPS) Wire Interoperability Protocol.

Because RTI Data Distribution Service **DDS::DataWriter** (p. 499) and **DDS::DataReader** (p. 433) cache Discovery information, inline QoS are largely redundant and thus unecessary. Only for other stateless implementations whose **DDS::DataReader** (p. 433) 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] false

6.27.3.2 System:: Boolean DDS::DataReaderProtocolQosPolicy::disable_positive_acks [get, set]

Whether the reader sends positive acknowledgements to writers.

If set to true, 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 false (the default), the reader will send ACKs to writers that expect ACKs (**DDS::DataWriterProtocolQosPolicy::disable_positive_acks** (p. 532) = false) and it will not send ACKs to writers that disable ACKs (**DDS::DataWriterProtocolQosPolicy::disable_positive_acks** (p. 532) = true)

[default] false

6.27.3.3 System:: Boolean DDS::DataReaderProtocolQosPolicy::propagate_dispose_of_unregistered_instances [get, set]

Indicates whether or not an instance can move to the **DDS::InstanceStateKind::NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 909) state without being in the **DDS::InstanceStateKind::ALIVE_INSTANCE_STATE** (p. 908) state.

This field only applies to keyed readers.

When the field is set to true, the **DataReader** (p. 433) will receive dispose notifications even if the instance is not alive.

To guarantee the key availability through the usage of the API **DDS::TypedDataReader::get_key_value** (p. 1365), this option should be used in combination **DDS::DataWriterProtocolQosPolicy::serialize_key_with_dispose** (p. 533) on the **DataWriter** (p. 499) that should be set to true.

[**default**] false

6.28 DDS::DataReaderProtocolStatus Struct Reference

*<<**eXtension**>>* *(p. 174)* The status of a reader's internal protocol related metrics, like the number of samples received, filtered, rejected; and status of wire protocol traffic.

`#include <managed_subscription.h>`

Public Attributes

- System::Int64 **received_sample_count**

  *The number of user samples from a remote **DataWriter** (p. 499) received for the first time by a local **DataReader** (p. 433).*

- System::Int64 **received_sample_count_change**

  *The incremental change in the number of user samples from a remote **DataWriter** (p. 499) received for the first time by a local **DataReader** (p. 433) since the last time the status was read.*

- System::Int64 **received_sample_bytes**

  *The number of bytes of user samples from a remote **DataWriter** (p. 499) received for the first time by a local **DataReader** (p. 433).*

- System::Int64 **received_sample_bytes_change**

  *The incremental change in the number of bytes of user samples from a remote **DataWriter** (p. 499) received for the first time by a local **DataReader** (p. 433) since the last time the status was read.*

- System::Int64 **duplicate_sample_count**

  *The number of samples from a remote **DataWriter** (p. 499) received, not for the first time, by a local **DataReader** (p. 433).*

- System::Int64 **duplicate_sample_count_change**

  *The incremental change in the number of samples from a remote **DataWriter** (p. 499) received, not for the first time, by a local **DataReader** (p. 433) since the last time the status was read.*

- System::Int64 **duplicate_sample_bytes**

  *The number of bytes of samples from a remote **DataWriter** (p. 499) received, not for the first time, by a local **DataReader** (p. 433).*

- System::Int64 **duplicate_sample_bytes_change**

*The incremental change in the number of bytes of samples from a remote **DataWriter** (p. 499) received, not for the first time, by a local **DataReader** (p. 433) since the last time the status was read.*

^ System::Int64 **filtered_sample_count**

*The number of user samples filtered by the local **DataReader** (p. 433) due to Content-Filtered Topics or Time-Based Filter.*

^ System::Int64 **filtered_sample_count_change**

*The incremental change in the number of user samples filtered by the local **DataReader** (p. 433) due to Content-Filtered Topics or Time-Based Filter since the last time the status was read.*

^ System::Int64 **filtered_sample_bytes**

*The number of bytes of user samples filtered by the local **DataReader** (p. 433) due to Content-Filtered Topics or Time-Based Filter.*

^ System::Int64 **filtered_sample_bytes_change**

*The incremental change in the number of bytes of user samples filtered by the local **DataReader** (p. 433) due to Content-Filtered Topics or Time-Based Filter since the last time the status was read.*

^ System::Int64 **received_heartbeat_count**

*The number of Heartbeats from a remote **DataWriter** (p. 499) received by a local **DataReader** (p. 433).*

^ System::Int64 **received_heartbeat_count_change**

*The incremental change in the number of Heartbeats from a remote **DataWriter** (p. 499) received by a local **DataReader** (p. 433) since the last time the status was read.*

^ System::Int64 **received_heartbeat_bytes**

*The number of bytes of Heartbeats from a remote **DataWriter** (p. 499) received by a local **DataReader** (p. 433).*

^ System::Int64 **received_heartbeat_bytes_change**

*The incremental change in the number of bytes of Heartbeats from a remote **DataWriter** (p. 499) received by a local **DataReader** (p. 433) since the last time the status was read.*

^ System::Int64 **sent_ack_count**

*The number of ACKs sent from a local **DataReader** (p. 433) to a matching remote **DataWriter** (p. 499).*

^ System::Int64 **sent_ack_count_change**

*The incremental change in the number of ACKs sent from a local **DataReader** (p. 433) to a matching remote **DataWriter** (p. 499) since the last time the status was read.*

^ System::Int64 **sent_ack_bytes**

*The number of bytes of ACKs sent from a local **DataReader** (p. 433) to a matching remote **DataWriter** (p. 499).*

^ System::Int64 **sent_ack_bytes_change**

*The incremental change in the number of bytes of ACKs sent from a local **DataReader** (p. 433) to a matching remote **DataWriter** (p. 499) since the last time the status was read.*

^ System::Int64 **sent_nack_count**

*The number of NACKs sent from a local **DataReader** (p. 433) to a matching remote **DataWriter** (p. 499).*

^ System::Int64 **sent_nack_count_change**

*The incremental change in the number of NACKs sent from a local **DataReader** (p. 433) to a matching remote **DataWriter** (p. 499) since the last time the status was read.*

^ System::Int64 **sent_nack_bytes**

*The number of bytes of NACKs sent from a local **DataReader** (p. 433) to a matching remote **DataWriter** (p. 499).*

^ System::Int64 **sent_nack_bytes_change**

*The incremental change in the number of bytes of NACKs sent from a local **DataReader** (p. 433) to a matching remote **DataWriter** (p. 499) since the last time the status was read.*

^ System::Int64 **received_gap_count**

*The number of GAPs received from remote **DataWriter** (p. 499) to this **DataReader** (p. 433).*

^ System::Int64 **received_gap_count_change**

*The incremental change in the number of GAPs received from remote **DataWriter** (p. 499) to this **DataReader** (p. 433) since the last time the status was read.*

^ System::Int64 **received_gap_bytes**

*The number of bytes of GAPs received from remote **DataWriter** (p. 499) to this **DataReader** (p. 433).*

^ System::Int64 **received_gap_bytes_change**

*The incremental change in the number of bytes of GAPs received from remote **DataWriter** (p. 499) to this **DataReader** (p. 433) since the last time the status was read.*

^ System::Int64 **rejected_sample_count**

The number of times a sample is rejected due to exceptions in the receive path.

^ System::Int64 **rejected_sample_count_change**

The incremental change in the number of times a sample is rejected due to exceptions in the receive path since the last time the status was read.

^ **SequenceNumber_t first_available_sample_sequence_number**

***Sequence** (p. 1163) number of the first available sample in a matched Datawriters reliability queue. Applicable only when retrieving matched **DataWriter** (p. 499) statuses.*

^ **SequenceNumber_t last_available_sample_sequence_number**

***Sequence** (p. 1163) number of the last available sample in a matched Datawriters reliability queue. Applicable only when retrieving matched **DataWriter** (p. 499) statuses.*

^ **SequenceNumber_t last_committed_sample_sequence_number**

***Sequence** (p. 1163) number of the newest sample received from the matched **DataWriter** (p. 499) committed to the DataReader's queue. Applicable only when retrieving matched **DataWriter** (p. 499) statuses.*

^ System::Int32 **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** (p. 499) statuses.*

6.28.1 Detailed Description

<<***eXtension***>> *(p. 174)* 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. 433)

6.28.2 Member Data Documentation

6.28.2.1 System::Int64 DDS::DataReaderProtocolStatus::received_sample_count

The number of user samples from a remote **DataWriter** (p. 499) received for the first time by a local **DataReader** (p. 433).

6.28.2.2 System::Int64 DDS::DataReaderProtocolStatus::received_sample_count_change

The incremental change in the number of user samples from a remote **DataWriter** (p. 499) received for the first time by a local **DataReader** (p. 433) since the last time the status was read.

6.28.2.3 System::Int64 DDS::DataReaderProtocolStatus::received_sample_bytes

The number of bytes of user samples from a remote **DataWriter** (p. 499) received for the first time by a local **DataReader** (p. 433).

6.28.2.4 System::Int64 DDS::DataReaderProtocolStatus::received_sample_bytes_change

The incremental change in the number of bytes of user samples from a remote **DataWriter** (p. 499) received for the first time by a local **DataReader** (p. 433) since the last time the status was read.

6.28.2.5 System::Int64 DDS::DataReaderProtocolStatus::duplicate_sample_count

The number of samples from a remote **DataWriter** (p. 499) received, not for the first time, by a local **DataReader** (p. 433).

Such samples can be redundant, out-of-order, etc. and are not stored in the reader's queue.

6.28.2.6 System::Int64 DDS::DataReaderProtocolStatus::duplicate_sample_count_change

The incremental change in the number of samples from a remote **DataWriter** (p. 499) received, not for the first time, by a local **DataReader** (p. 433) 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 System::Int64 DDS::DataReaderProtocolStatus::duplicate_sample_bytes

The number of bytes of samples from a remote **DataWriter** (p. 499) received, not for the first time, by a local **DataReader** (p. 433).

Such samples can be redundant, out-of-order, etc. and are not stored in the reader's queue.

6.28.2.8 System::Int64 DDS::DataReaderProtocolStatus::duplicate_sample_bytes_change

The incremental change in the number of bytes of samples from a remote **DataWriter** (p. 499) received, not for the first time, by a local **DataReader** (p. 433) 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 System::Int64 DDS::DataReaderProtocolStatus::filtered_sample_count

The number of user samples filtered by the local **DataReader** (p. 433) due to Content-Filtered Topics or Time-Based Filter.

6.28.2.10 System::Int64 DDS::DataReaderProtocolStatus::filtered_sample_count_change

The incremental change in the number of user samples filtered by the local **DataReader** (p. 433) due to Content-Filtered Topics or Time-Based Filter since the last time the status was read.

6.28.2.11 System::Int64 DDS::DataReaderProtocolStatus::filtered_sample_bytes

The number of bytes of user samples filtered by the local **DataReader** (p. 433) due to Content-Filtered Topics or Time-Based Filter.

6.28.2.12 System::Int64 DDS::DataReaderProtocolStatus::filtered_sample_bytes_change

The incremental change in the number of bytes of user samples filtered by the local **DataReader** (p. 433) due to Content-Filtered Topics or Time-Based Filter since the last time the status was read.

6.28.2.13 System::Int64 DDS::DataReaderProtocolStatus::received_heartbeat_count

The number of Heartbeats from a remote **DataWriter** (p. 499) received by a local **DataReader** (p. 433).

6.28.2.14 System::Int64 DDS::DataReaderProtocolStatus::received_heartbeat_count_change

The incremental change in the number of Heartbeats from a remote **DataWriter** (p. 499) received by a local **DataReader** (p. 433) since the last time the status was read.

6.28.2.15 System::Int64 DDS::DataReaderProtocolStatus::received_heartbeat_bytes

The number of bytes of Heartbeats from a remote **DataWriter** (p. 499) received by a local **DataReader** (p. 433).

6.28.2.16 System::Int64 DDS::DataReaderProtocolStatus::received_heartbeat_bytes_change

The incremental change in the number of bytes of Heartbeats from a remote **DataWriter** (p. 499) received by a local **DataReader** (p. 433) since the last time the status was read.

6.28.2.17 System::Int64 DDS::DataReaderProtocolStatus::sent_ack_count

The number of ACKs sent from a local **DataReader** (p. 433) to a matching remote **DataWriter** (p. 499).

6.28.2.18 System::Int64 DDS::DataReaderProtocolStatus::sent_ack_count_change

The incremental change in the number of ACKs sent from a local **DataReader** (p. 433) to a matching remote **DataWriter** (p. 499) since the last time the status was read.

6.28.2.19 System::Int64 DDS::DataReaderProtocolStatus::sent_ack_bytes

The number of bytes of ACKs sent from a local **DataReader** (p. 433) to a matching remote **DataWriter** (p. 499).

6.28.2.20 System::Int64 DDS::DataReaderProtocolStatus::sent_ack_bytes_change

The incremental change in the number of bytes of ACKs sent from a local **DataReader** (p. 433) to a matching remote **DataWriter** (p. 499) since the last time the status was read.

6.28.2.21 System::Int64 DDS::DataReaderProtocolStatus::sent_nack_count

The number of NACKs sent from a local **DataReader** (p. 433) to a matching remote **DataWriter** (p. 499).

6.28.2.22 System::Int64 DDS::DataReaderProtocolStatus::sent_nack_count_change

The incremental change in the number of NACKs sent from a local **DataReader** (p. 433) to a matching remote **DataWriter** (p. 499) since the last time the status was read.

6.28.2.23 System::Int64 DDS::DataReaderProtocolStatus::sent_nack_bytes

The number of bytes of NACKs sent from a local **DataReader** (p. 433) to a matching remote **DataWriter** (p. 499).

6.28.2.24 System::Int64 DDS::DataReaderProtocolStatus::sent_nack_bytes_change

The incremental change in the number of bytes of NACKs sent from a local **DataReader** (p. 433) to a matching remote **DataWriter** (p. 499) since the last time the status was read.

6.28.2.25 System::Int64 DDS::DataReaderProtocolStatus::received_gap_count

The number of GAPs received from remote **DataWriter** (p. 499) to this **DataReader** (p. 433).

6.28.2.26 System::Int64 DDS::DataReaderProtocolStatus::received_gap_count_change

The incremental change in the number of GAPs received from remote **DataWriter** (p. 499) to this **DataReader** (p. 433) since the last time the status was read.

6.28.2.27 System::Int64 DDS::DataReaderProtocolStatus::received_gap_bytes

The number of bytes of GAPs received from remote **DataWriter** (p. 499) to this **DataReader** (p. 433).

6.28.2.28 System::Int64 DDS::DataReaderProtocolStatus::received_gap_bytes_change

The incremental change in the number of bytes of GAPs received from remote **DataWriter** (p. 499) to this **DataReader** (p. 433) since the last time the status was read.

6.28.2.29 System::Int64 DDS::DataReaderProtocolStatus::rejected_sample_count

The number of times a sample is rejected due to exceptions in the receive path.

6.28.2.30 System::Int64 DDS::DataReaderProtocolStatus::rejected_sample_count_change

The incremental change in the number of times a sample is rejected due to exceptions in the receive path since the last time the status was read.

6.28.2.31 SequenceNumber_t DDS::DataReaderProtocolStatus::first_available_sample_sequence_number

Sequence (p. 1163) number of the first available sample in a matched Datawriters reliability queue. Applicable only when retrieving matched **DataWriter** (p. 499) statuses.

6.28.2.32 SequenceNumber_t DDS::DataReaderProtocolStatus::last_available_sample_sequence_number

Sequence (p. 1163) number of the last available sample in a matched Datawriters reliability queue. Applicable only when retrieving matched **DataWriter** (p. 499) statuses.

6.28.2.33 SequenceNumber_t DDS::DataReaderProtocolStatus::last_committed_sample_sequence_number

Sequence (p. 1163) number of the newest sample received from the matched **DataWriter** (p. 499) committed to the DataReader's queue. Applicable only when retrieving matched **DataWriter** (p. 499) statuses.

6.28.2.34 System::Int32 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** (p. 499) statuses.

6.29 DDS::DataReaderQos Class Reference

QoS policies supported by a **DDS::DataReader** (p. 433) entity.

`#include <managed_subscription.h>`

Public Attributes

- ^ **DurabilityQosPolicy durability**
  Durability policy, ***DURABILITY*** *(p. 276).*
- ^ **DeadlineQosPolicy deadline**
  Deadline policy, ***DEADLINE*** *(p. 281).*
- ^ **LatencyBudgetQosPolicy latency_budget**
  Latency budget policy, ***LATENCY_BUDGET*** *(p. 282).*
- ^ **LivelinessQosPolicy liveliness**
  Liveliness policy, ***LIVELINESS*** *(p. 286).*
- ^ **ReliabilityQosPolicy reliability**
  Reliability policy, ***RELIABILITY*** *(p. 290).*
- ^ **DestinationOrderQosPolicy destination_order**
  Destination order policy, ***DESTINATION_ORDER*** *(p. 292).*
- ^ **HistoryQosPolicy history**
  History policy, ***HISTORY*** *(p. 294).*
- ^ **ResourceLimitsQosPolicy resource_limits**
  Resource limits policy, ***RESOURCE_LIMITS*** *(p. 298).*
- ^ **UserDataQosPolicy^ user_data**
  User data policy, ***USER_DATA*** *(p. 273).*
- ^ **OwnershipQosPolicy ownership**
  Ownership policy, ***OWNERSHIP*** *(p. 283).*
- ^ **TimeBasedFilterQosPolicy time_based_filter**
  Time-based filter policy, ***TIME_BASED_FILTER*** *(p. 288).*
- ^ **ReaderDataLifecycleQosPolicy reader_data_lifecycle**
  Reader data lifecycle policy, ***READER_DATA_LIFECYCLE*** *(p. 303).*

- **DataReaderResourceLimitsQosPolicy reader_resource_limits**

  <<**eXtension**>> (p. 174) ***DDS::DataReader*** *(p. 433) resource limits policy,* ***DATA_READER_RESOURCE_LIMITS*** *(p. 331). This policy is an extension to the DDS standard.*

- **DataReaderProtocolQosPolicy protocol**

  <<**eXtension**>> (p. 174) ***DDS::DataReader*** *(p. 433) protocol policy,* ***DATA_READER_PROTOCOL*** *(p. 337)*

- **TransportSelectionQosPolicy^ transport_selection**

  <<**eXtension**>> (p. 174) *Transport selection policy,* ***TRANSPORT_SELECTION*** *(p. 308).*

- **TransportUnicastQosPolicy^ unicast**

  <<**eXtension**>> (p. 174) *Unicast transport policy,* ***TRANSPORT_UNICAST*** *(p. 309).*

- **TransportMulticastQosPolicy^ multicast**

  <<**eXtension**>> (p. 174) *Multicast transport policy,* ***TRANSPORT_MULTICAST*** *(p. 310).*

- **PropertyQosPolicy^ property_qos**

  <<**eXtension**>> (p. 174) *Property policy,* ***PROPERTY*** *(p. 357).*

- **TypeSupportQosPolicy type_support**

  <<**eXtension**>> (p. 174) *type support data,* ***TYPESUPPORT*** *(p. 350).*

6.29.1 Detailed Description

QoS policies supported by a **DDS::DataReader** (p. 433) entity.

You must set certain members in a consistent manner:

DDS::DataReaderQos::deadline.period >= DDS::DataReaderQos::time_based_filter.minimum_separation

DDS::DataReaderQos::history.depth <= DDS::DataReaderQos::resource_limits.max_samples_per_instance

DDS::DataReaderQos::resource_limits.max_samples_per_instance <= DDS::DataReaderQos::resource_limits.max_samples DDS::DataReaderQos::resource_limits.initial_samples <= DDS::DataReaderQos::resource_limits.max_samples

DDS::DataReaderQos::resource_limits.initial_instances <= DDS::DataReaderQos::resource_limits.max_instances

DDS::DataReaderQos::reader_resource_limits.initial_remote_writers_per_instance <= DDS::DataReaderQos::reader_resource_limits.max_remote_writers_per_instance

DDS::DataReaderQos::reader_resource_limits.initial_infos <= DDS::DataReaderQos::reader_resource_limits.max_infos

DDS::DataReaderQos::reader_resource_limits.max_remote_writers_per_instance <= DDS::DataReaderQos::reader_resource_limits.max_remote_writers

DDS::DataReaderQos::reader_resource_limits.max_samples_per_remote_writer <= DDS::DataReaderQos::resource_limits.max_samples

length of DDS::DataReaderQos::user_data.value <= DDS::DomainParticipantQos::resource_limits.reader_user_data_max_length

If any of the above are not true, **DDS::DataReader::set_qos** (p. 448) and **DDS::DataReader::set_qos_with_profile** (p. 448) will fail with **DDS::Retcode_InconsistentPolicy** (p. 1119)

6.29.2 Member Data Documentation

6.29.2.1 DurabilityQosPolicy DDS::DataReaderQos::durability

Durability policy, **DURABILITY** (p. 276).

6.29.2.2 DeadlineQosPolicy DDS::DataReaderQos::deadline

Deadline policy, **DEADLINE** (p. 281).

6.29.2.3 LatencyBudgetQosPolicy DDS::DataReaderQos::latency_budget

Latency budget policy, **LATENCY_BUDGET** (p. 282).

6.29.2.4 LivelinessQosPolicy DDS::DataReaderQos::liveliness

Liveliness policy, **LIVELINESS** (p. 286).

6.29.2.5 ReliabilityQosPolicy DDS::DataReaderQos::reliability

Reliability policy, **RELIABILITY** (p. 290).

6.29.2.6 DestinationOrderQosPolicy DDS::DataReaderQos::destination_order

Destination order policy, **DESTINATION_ORDER** (p. 292).

6.29.2.7 HistoryQosPolicy DDS::DataReaderQos::history

History policy, **HISTORY** (p. 294).

6.29.2.8 ResourceLimitsQosPolicy DDS::DataReaderQos::resource_limits

Resource limits policy, **RESOURCE_LIMITS** (p. 298).

6.29.2.9 UserDataQosPolicy ^ DDS::DataReaderQos::user_data

User data policy, **USER_DATA** (p. 273).

6.29.2.10 OwnershipQosPolicy DDS::DataReaderQos::ownership

Ownership policy, **OWNERSHIP** (p. 283).

6.29.2.11 TimeBasedFilterQosPolicy DDS::DataReaderQos::time_based_filter

Time-based filter policy, **TIME_BASED_FILTER** (p. 288).

6.29.2.12 ReaderDataLifecycleQosPolicy DDS::DataReaderQos::reader_data_lifecycle

Reader data lifecycle policy, **READER_DATA_LIFECYCLE** (p. 303).

6.29.2.13 DataReaderResourceLimitsQosPolicy DDS::DataReaderQos::reader_resource_limits

<<*eXtension*>> *(p. 174)* **DDS::DataReader** (p. 433) resource limits policy, **DATA_READER_RESOURCE_LIMITS** (p. 331). This policy is an extension to the DDS standard.

6.29.2.14 DataReaderProtocolQosPolicy DDS::DataReaderQos::protocol

<<*eXtension*>> *(p. 174)* **DDS::DataReader** (p. 433) protocol policy, **DATA_READER_PROTOCOL** (p. 337)

6.29.2.15 TransportSelectionQosPolicy ^ DDS::DataReaderQos::transport_selection

<<*eXtension*>> *(p. 174)* Transport selection policy, **TRANSPORT_SELECTION** (p. 308).

Specifies the transports available for use by the **DDS::DataReader** (p. 433).

6.29.2.16 TransportUnicastQosPolicy ^ DDS::DataReaderQos::unicast

<<*eXtension*>> *(p. 174)* Unicast transport policy, **TRANSPORT_UNICAST** (p. 309).

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. 499) entities in the domain.

6.29.2.17 TransportMulticastQosPolicy ^ DDS::DataReaderQos::multicast

<<*eXtension*>> *(p. 174)* Multicast transport policy, **TRANSPORT_MULTICAST** (p. 310).

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. 499) entities in the domain.

6.29.2.18 PropertyQosPolicy ^ DDS::DataReaderQos::property_qos

<<*eXtension*>> *(p. 174)* Property policy, **PROPERTY** (p. 357).

6.29.2.19 TypeSupportQosPolicy DDS::DataReaderQos::type_support

<<*eXtension*>> *(p. 174)* type support data, **TYPESUPPORT** (p. 350).

Optional value that is passed to a type plugin's on_endpoint_attached and deserialization functions.

6.30 DDS::DataReaderResourceLimitsQosPolicy Struct Reference

Various settings that configure how a **DDS::DataReader** (p. 433) allocates and uses physical memory for internal resources.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_datareaderresourcelimits_qos_policy_name** ()

  *Stringified human-readable name for **DDS::DataReaderResourceLimitsQosPolicy** (p. 486).*

Public Attributes

- System::Int32 **max_remote_writers**

  *The maximum number of remote writers from which a **DDS::DataReader** (p. 433) may read, including all instances.*

- System::Int32 **max_remote_writers_per_instance**

  *The maximum number of remote writers from which a **DDS::DataReader** (p. 433) may read a single instance.*

- System::Int32 **max_samples_per_remote_writer**

  *The maximum number of out-of-order samples from a given remote **DDS::DataWriter** (p. 499) that a **DDS::DataReader** (p. 433) may store when maintaining a reliable connection to the **DDS::DataWriter** (p. 499).*

- System::Int32 **max_infos**

  *The maximum number of info units that a **DDS::DataReader** (p. 433) can use to store **DDS::SampleInfo** (p. 1148).*

- System::Int32 **initial_remote_writers**

  *The initial number of remote writers from which a **DDS::DataReader** (p. 433) may read, including all instances.*

- System::Int32 **initial_remote_writers_per_instance**

  *The initial number of remote writers from which a **DDS::DataReader** (p. 433) may read a single instance.*

^ System::Int32 **initial_infos**

*The initial number of info units that a **DDS::DataReader** (p. 433) can have, which are used to store **DDS::SampleInfo** (p. 1148).*

^ System::Int32 **initial_outstanding_reads**

*The initial number of outstanding calls to read/take (or one of their variants) on the same **DDS::DataReader** (p. 433) for which memory has not been returned by calling **DDS::TypedDataReader::return_loan** (p. 1364).*

^ System::Int32 **max_outstanding_reads**

*The maximum number of outstanding read/take calls (or one of their variants) on the same **DDS::DataReader** (p. 433) for which memory has not been returned by calling **DDS::TypedDataReader::return_loan** (p. 1364).*

^ System::Int32 **max_samples_per_read**

*The maximum number of data samples that the application can receive from the middleware in a single call to **DDS::TypedDataReader::read** (p. 1341) or **DDS::TypedDataReader::take** (p. 1342). If more data exists in the middleware, the application will need to issue multiple read/take calls.*

^ System::Int32 **max_fragmented_samples**

*The maximum number of samples for which the **DDS::DataReader** (p. 433) may store fragments at a given point in time.*

^ System::Int32 **initial_fragmented_samples**

*The initial number of samples for which a **DDS::DataReader** (p. 433) may store fragments.*

^ System::Int32 **max_fragmented_samples_per_remote_writer**

*The maximum number of samples per remote writer for which a **DDS::DataReader** (p. 433) may store fragments.*

^ System::Int32 **max_fragments_per_sample**

Maximum number of fragments for a single sample.

^ System::Int32 **max_total_instances**

*Maximum number of instances for which a **DataReader** (p. 433) will keep state.*

^ System::Int32 **max_remote_virtual_writers_per_instance**

The maximum number of virtual remote writers that can be associated with an instance.

^ System::Int32 **initial_remote_virtual_writers_per_instance**

The initial number of virtual remote writers per instance.

^ System::Int32 **max_query_condition_filters**

The maximum number of query condition filters a reader is allowed.

Properties

^ System::Boolean **disable_fragmentation_support** `[get, set]`

*Determines whether the **DDS::DataReader** (p. 433) can receive fragmented samples.*

^ System::Boolean **dynamically_allocate_fragmented_samples** `[get, set]`

*Determines whether the **DDS::DataReader** (p. 433) pre-allocates storage for storing fragmented samples.*

^ static System::Int32 **AUTO_MAX_TOTAL_INSTANCES** `[get]`

<<**eXtension**>> (p. 174) *This value is used to make **DDS::DataReaderResourceLimitsQosPolicy::max_total_instances** (p. 494) equal to **DDS::ResourceLimitsQosPolicy::max_instances** (p. 1112).*

6.30.1 Detailed Description

Various settings that configure how a **DDS::DataReader** (p. 433) 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. 433) 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 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::DataReader** (p. 433). By setting the initial size to the maximum size, you will prevent RTI Data Distribution Service from dynamically allocating any memory after the creation of the **DDS::DataReader** (p. 433).

This QoS policy is an extension to the DDS standard.

Entity:

DDS::DataReader (p. 433)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)

6.30.2 Member Data Documentation

6.30.2.1 System::Int32 DDS::DataReaderResourceLimitsQosPolicy::max_remote_writers

The maximum number of remote writers from which a **DDS::DataReader** (p. 433) may read, including all instances.

[**default**] DDS::LENGTH_UNLIMITED

[**range**] [1, 1 million] or DDS::LENGTH_UNLIMITED, >= 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.

Note: For efficiency, set `max_remote_writers` >= **DDS::DataReaderResourceLimitsQosPolicy::max_remote_writers_per_instance** (p. 489).

6.30.2.2 System::Int32 DDS::DataReaderResourceLimitsQosPolicy::max_remote_writers_per_instance

The maximum number of remote writers from which a **DDS::DataReader** (p. 433) may read a single instance.

[**default**] DDS::LENGTH_UNLIMITED

[**range**] [1, 1024] or DDS::LENGTH_UNLIMITED, <= max_remote_writers or DDS::LENGTH_UNLIMITED, >= 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.

Note: For efficiency, set `max_remote_writers_per_instance` <= **DDS::DataReaderResourceLimitsQosPolicy::max_remote_writers** (p. 489)

6.30.2.3 System::Int32 DDS::DataReaderResourceLimitsQosPolicy::max_samples_per_remote_writer

The maximum number of out-of-order samples from a given remote **DDS::DataWriter** (p. 499) that a **DDS::DataReader** (p. 433) may store when maintaining a reliable connection to the **DDS::DataWriter** (p. 499).

[default] DDS::LENGTH_UNLIMITED

[range] [1, 100 million] or DDS::LENGTH_UNLIMITED, <= **DDS::ResourceLimitsQosPolicy::max_samples** (p. 1112)

6.30.2.4 System::Int32 DDS::DataReaderResourceLimitsQosPolicy::max_infos

The maximum number of info units that a **DDS::DataReader** (p. 433) can use to store **DDS::SampleInfo** (p. 1148).

When read/take is called on a data reader, the data reader 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

[range] [1, 1 million] or DDS::LENGTH_UNLIMITED, >= initial_infos

6.30.2.5 System::Int32 DDS::DataReaderResourceLimitsQosPolicy::initial_remote_writers

The initial number of remote writers from which a **DDS::DataReader** (p. 433) 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. 491).

6.30.2.6 System::Int32 DDS::DataReaderResourceLimitsQosPolicy::initial_remote_writers_per_instance

The initial number of remote writers from which a **DDS::DataReader** (p. 433) 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. 490).

6.30.2.7 System::Int32 DDS::DataReaderResourceLimitsQosPolicy::initial_infos

The initial number of info units that a **DDS::DataReader** (p. 433) can have, which are used to store **DDS::SampleInfo** (p. 1148).

[default] 32

[range] [1,1 million], <= max_infos

6.30.2.8 System::Int32 DDS::DataReaderResourceLimitsQosPolicy::initial_outstanding_reads

The initial number of outstanding calls to read/take (or one of their variants) on the same **DDS::DataReader** (p. 433) for which memory has not been returned by calling **DDS::TypedDataReader::return_loan** (p. 1364).

[default] 2

[range] [1, 65536], <= max_outstanding_reads

6.30.2.9 System::Int32 DDS::DataReaderResourceLimitsQosPolicy::max_outstanding_reads

The maximum number of outstanding read/take calls (or one of their variants) on the same **DDS::DataReader** (p. 433) for which memory has not been returned by calling **DDS::TypedDataReader::return_loan** (p. 1364).

[default] DDS::LENGTH_UNLIMITED

[range] [1, 65536] or DDS::LENGTH_UNLIMITED, >= initial_outstanding_reads

6.30.2.10 System::Int32 DDS::DataReaderResourceLimitsQosPolicy::max_samples_per_read

The maximum number of data samples that the application can receive from the middleware in a single call to **DDS::TypedDataReader::read** (p. 1341) or **DDS::TypedDataReader::take** (p. 1342). 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. 376) for more information about this feature.) When polling for data or using a **DDS::WaitSet** (p. 1411), 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]

6.30.2.11 System::Int32 DDS::DataReaderResourceLimitsQosPolicy::max_fragmented_samples

The maximum number of samples for which the **DDS::DataReader** (p. 433) may store fragments at a given point in time.

At any given time, a **DDS::DataReader** (p. 433) 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. 499) 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. 1112).

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. 496) is false.

[**default**] 1024

[**range**] [1, 1 million]

6.30.2.12 System::Int32 DDS::DataReaderResourceLimitsQosPolicy::initial_fragmented_samples

The initial number of samples for which a **DDS::DataReader** (p. 433) may store fragments.

Only applies if **DDS::DataReaderResourceLimitsQosPolicy::disable_fragmentation_support** (p. 496) is false.

[**default**] 4

[**range**] [1,1024], <= max_fragmented_samples

6.30.2.13 System::Int32 DDS::DataReaderResourceLimitsQosPolicy::max_fragmented_samples_per_remote_writer

The maximum number of samples per remote writer for which a **DDS::DataReader** (p. 433) may store fragments.

Logical limit so a single remote writer cannot consume all available resources.

Only applies if **DDS::DataReaderResourceLimitsQosPolicy::disable_fragmentation_support** (p. 496) is false.

[**default**] 256

[**range**] [1, 1 million], <= max_fragmented_samples

6.30.2.14 System::Int32 DDS::DataReaderResourceLimitsQosPolicy::max_fragments_per_sample

Maximum number of fragments for a single sample.

Only applies if **DDS::DataReaderResourceLimitsQosPolicy::disable_fragmentation_support** (p. 496) is false.

[**default**] 512

[**range**] [1, 1 million] or DDS::LENGTH_UNLIMITED

6.30.2.15 System::Int32 DDS::DataReaderResourceLimitsQosPolicy::max_total_instances

Maximum number of instances for which a **DataReader** (p. 433) will keep state.

The maximum number of instances actively managed by a **DataReader** (p. 433) is determined by **DDS::ResourceLimitsQosPolicy::max_instances** (p. 1112).

These instances have associated DataWriters or samples in the DataReader's queue and are visible to the user through operations such as **DDS::TypedDataReader::take** (p. 1342), **DDS::TypedDataReader::read** (p. 1341), and **DDS::TypedDataReader::get_key_value** (p. 1365).

The features Durable Reader State, MultiChannel DataWriters and RTI Persistence Service require RTI Data Distribution Service 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** (p. 499) 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 Data Distribution Service will check the resource limit **DDS::ResourceLimitsQosPolicy::max_instances** (p. 1112). If the limit is exceeded, RTI Data Distribution Service will drop the sample and report it as lost and rejected. If the limit is not exceeded, RTI Data Distribution Service will check max_total_instances. If max_total_instances is exceeded, RTI Data Distribution Service 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.

[default] **DDS::DataReaderResourceLimitsQosPolicy::AUTO_MAX_TOTAL_INSTANCES** (p. 332)

[range] [1, 1 million] or DDS::LENGTH_UNLIMITED or **DDS::DataReaderResourceLimitsQosPolicy::AUTO_MAX_TOTAL_INSTANCES** (p. 332), >= **DDS::ResourceLimitsQosPolicy::max_instances** (p. 1112)

6.30.2.16 System::Int32 DDS::DataReaderResourceLimitsQosPolicy::max_remote_virtual_writers_per_instance

The maximum number of virtual remote writers that can be associated with an instance.

[**default**] DDS::LENGTH_UNLIMITED

[**range**] [1, 1024] or DDS::LENGTH_UNLIMITED, >= 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 Data Distribution Service 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** (p. 499) 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 Data Distribution Service 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.

6.30.2.17 System::Int32 DDS::DataReaderResourceLimitsQosPolicy::initial_remote_virtual_writers_per_instance

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.

6.30.2.18 System::Int32 DDS::DataReaderResourceLimitsQosPolicy::max_query_condition_filters

The maximum number of query condition filters a reader is allowed.

[default] 4

[range] [0, 32]

This value determines the maximum number of unique query condition content filters that a reader may create.

Each query condition content filter is comprised of both its `query_expression` and `query_parameters`. Two query conditions that have the same `query_expression` will require unique query condition filters if their `query_paramters` differ. Query conditions that differ only in their state masks will share the same query condition filter.

6.30.3 Property Documentation

6.30.3.1 System:: Boolean DDS::DataReaderResourceLimitsQosPolicy::disable_fragmentation_support [get, set]

Determines whether the **DDS::DataReader** (p. 433) can receive fragmented samples.

When fragmentation support is not needed, disabling fragmentation support will save some memory resources.

[default] false

6.30.3.2 System:: Boolean DDS::DataReaderResourceLimitsQosPolicy::dynamically_allocate_fragmented_samples [get, set]

Determines whether the **DDS::DataReader** (p. 433) pre-allocates storage for storing fragmented samples.

By default, the middleware will allocate memory upfront for storing fragments for up to **DDS::DataReaderResourceLimitsQosPolicy::initial_fragmented_samples** (p. 493) samples. This memory may grow up to **DDS::DataReaderResourceLimitsQosPolicy::max_fragmented_samples** (p. 492) if needed.

If `dynamically_allocate_fragmented_samples` is set to true, 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 may be 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 run-time.

Only applies if **DDS::DataReaderResourceLimitsQosPolicy::disable_fragmentation_support** (p. 496) is false.

[**default**] false

6.31 DDS::DataReaderSeq Class Reference

Declares IDL `sequence` < **DDS::DataReader** (p. 433) > .

`#include <managed_subscription.h>`

Inheritance diagram for DDS::DataReaderSeq::

6.31.1 Detailed Description

Declares IDL `sequence` < **DDS::DataReader** (p. 433) > .

See also:

DDS::Sequence (p. 1163)

6.32 DDS::DataWriter Class Reference

<<***interface***>> *(p. 175)* Allows an application to set the value of the data to be published under a given **DDS::Topic** (p. 1258).

`#include <managed_publication.h>`

Inheritance diagram for DDS::DataWriter::

Public Member Functions

- ^ void **get_liveliness_lost_status** (**LivelinessLostStatus**% status)

  Accesses the DDS::StatusKind::LIVELINESS_LOST_STATUS communication status.

- ^ void **get_offered_deadline_missed_status** (**OfferedDeadlineMissedStatus**% status)

  Accesses the DDS::StatusKind::OFFERED_DEADLINE_MISSED_STATUS communication status.

- ^ void **get_offered_incompatible_qos_status** (**OfferedIncompatibleQosStatus**^ status)

  Accesses the DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS communication status.

- ^ void **get_publication_matched_status** (**PublicationMatchedStatus**% status)

  Accesses the DDS::StatusKind::PUBLICATION_MATCHED_STATUS communication status.

- ^ void **get_reliable_writer_cache_changed_status** (**ReliableWriterCacheChangedStatus**% status)

  <<**eXtension**>> (p. 174) *Get the reliable cache status for this writer.*

- ^ void **get_reliable_reader_activity_changed_status** (**ReliableReaderActivityChangedStatus**% status)

  <<**eXtension**>> (p. 174) *Get the reliable reader activity changed status for this writer.*

- ^ virtual void **get_datawriter_cache_status** (**DataWriterCacheStatus**% status)

  <<**eXtension**>> (p. 174) *Get the datawriter cache status for this writer.*

^ virtual void **get_datawriter_protocol_status** (**DataWriterProtocolStatus**% status)

<<**eXtension**>> (p. 174) *Get the datawriter protocol status for this writer.*

^ virtual void **get_matched_subscription_datawriter_protocol_status** (**DataWriterProtocolStatus**% status, **InstanceHandle_t**% subscription_handle)

<<**eXtension**>> (p. 174) *Get the datawriter protocol status for this writer, per matched subscription identified by the subscription_handle.*

^ virtual void **get_matched_subscription_datawriter_protocol_status_by_locator** (**DataWriterProtocolStatus**% status, **Locator_t**^ locator)

<<**eXtension**>> (p. 174) *Get the datawriter protocol status for this writer, per matched subscription identified by the locator.*

^ void **assert_liveliness** ()

*This operation manually asserts the liveliness of this **DDS::DataWriter** (p. 499).*

^ virtual void **get_matched_subscription_locators** (**LocatorSeq**^ locators)

<<**eXtension**>> (p. 174) *Retrieve the list of locators for subscriptions currently "associated" with this **DDS::DataWriter** (p. 499).*

^ void **get_matched_subscriptions** (**InstanceHandleSeq**^ subscription_handles)

*Retrieve the list of subscriptions currently "associated" with this **DDS::DataWriter** (p. 499).*

^ void **get_matched_subscription_data** (**SubscriptionBuiltinTopicData**^ subscription_data, **InstanceHandle_t**% subscription_handle)

*This operation retrieves the information on a subscription that is currently "associated" with the **DDS::DataWriter** (p. 499).*

^ **Topic**^ **get_topic** ()

*This operation returns the **DDS::Topic** (p. 1258) associated with the **DDS::DataWriter** (p. 499).*

^ **Publisher**^ **get_publisher** ()

*This operation returns the **DDS::Publisher** (p. 1044) to which the **DDS::DataWriter** (p. 499) belongs.*

^ void **wait_for_acknowledgments** (**Duration_t** max_wait)

*Blocks the calling thread until all data written by reliable **DDS::DataWriter** (p. 499) entity is acknowledged, or until timeout expires.*

^ void **wait_for_asynchronous_publishing** (**Duration_t** max_wait)

<<**eXtension**>> (p. 174) *Blocks the calling thread until asynchronous sending is complete.*

^ void **set_qos** (**DataWriterQos**^ qos)

Sets the writer QoS.

^ void **set_qos_with_profile** (System::String^ library_name, System::String^ profile_name)

<<**eXtension**>> (p. 174) *Change the QoS of this writer using the input XML QoS profile.*

^ void **get_qos** (**DataWriterQos**^ qos)

Gets the writer QoS.

^ void **set_listener** (**DataWriterListener**^ l, **StatusMask** mask)

Sets the writer listener.

^ **DataWriterListener**^ **get_listener** ()

Get the writer listener.

^ void **flush** ()

<<**eXtension**>> (p. 174) *Flushes the batch in progress in the context of the calling thread.*

^ **InstanceHandle_t** **register_instance_untyped** (System::Object^ instance_data)

Register a new instance with this writer.

^ **InstanceHandle_t** **register_instance_w_timestamp_untyped** (System::Object^ instance_data, **DDS::Time_t**% source_timestamp)

Register a new instance with this writer using the given time instead of the current time.

^ void **unregister_instance_untyped** (System::Object^ instance_data, **DDS::InstanceHandle_t**% handle)

Unregister a new instance from this writer.

^ void **unregister_instance_w_timestamp_untyped** (System::Object^ instance_data, **DDS::InstanceHandle_t**% handle, **DDS::Time_t**% source_timestamp)

Unregister a new instance from this writer using the given time instead of the current time.

^ void **write_untyped** (System::Object^ instance_data, **DDS::InstanceHandle_t**% handle)

Publish a data sample.

^ void **write_w_timestamp_untyped** (System::Object^ instance_data, **DDS::InstanceHandle_t**% handle, **DDS::Time_t**% source_timestamp)

Publish a data sample using the given time instead of the current time.

^ void **dispose_untyped** (System::Object^ instance_data, **DDS::InstanceHandle_t**% instance_handle)

Dispose a data sample.

^ void **dispose_w_timestamp_untyped** (System::Object^ instance_data, **DDS::InstanceHandle_t**% instance_handle, **DDS::Time_t**% source_timestamp)

Dispose a data sample using the given time instead of the current time.

^ void **get_key_value_untyped** (System::Object^ key_holder, **DDS::InstanceHandle_t**% handle)

Fill in the key fields of the given data sample.

^ **InstanceHandle_t lookup_instance_untyped** (System::Object^ key_holder)

Given a sample with the given key field values, return the handle corresponding to its instance.

^ virtual void **enable** () override

*Enables the **DDS::Entity*** (p. 845).

^ virtual **StatusCondition**^ **get_statuscondition** () override

*Allows access to the **DDS::StatusCondition*** (p. 1183) *associated with the **DDS::Entity*** (p. 845).

^ virtual **StatusMask get_status_changes** () override

*Retrieves the list of communication statuses in the **DDS::Entity*** (p. 845) *that are triggered.*

^ virtual **InstanceHandle_t get_instance_handle** () override

*Allows access to the **DDS::InstanceHandle_t*** (p. 905) *associated with the **DDS::Entity*** (p. 845).

6.32.1 Detailed Description

<<*interface*>> *(p. 175)* Allows an application to set the value of the data to be published under a given **DDS::Topic** (p. 1258).

QoS:

DDS::DataWriterQos (p. 546)

Status:

DDS::StatusKind::LIVELINESS_LOST_STATUS, **DDS::LivelinessLostStatus** (p. 958); DDS::StatusKind::OFFERED_DEADLINE_MISSED_STATUS, **DDS::OfferedDeadlineMissedStatus** (p. 989); DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS, **DDS::OfferedIncompatibleQosStatus** (p. 991); DDS::StatusKind::PUBLICATION_MATCHED_STATUS, **DDS::PublicationMatchedStatus** (p. 1041); DDS::StatusKind::RELIABLE_READER_ACTIVITY_CHANGED_STATUS, **DDS::ReliableReaderActivityChangedStatus** (p. 1098); DDS::StatusKind::RELIABLE_WRITER_CACHE_CHANGED_STATUS, DDS::ReliableWriterCacheChangedStatus

Listener:

DDS::DataWriterListener (p. 524)

A **DDS::DataWriter** (p. 499) is attached to exactly one **DDS::Publisher** (p. 1044), that acts as a factory for it.

A **DDS::DataWriter** (p. 499) is bound to exactly one **DDS::Topic** (p. 1258) and therefore to exactly one data type. The **DDS::Topic** (p. 1258) must exist prior to the **DDS::DataWriter** (p. 499)'s creation.

DDS::DataWriter (p. 499) is an abstract class. It must be specialized for each particular application data-type (see **USER_DATA** (p. 273)). The additional methods or functions that must be defined in the auto-generated class for a hypothetical application type `Foo` (p. 877) are specified in the example type **DDS::DataWriter** (p. 499).

The following operations may be called even if the **DDS::DataWriter** (p. 499) is not enabled. Other operations will fail with **DDS::Retcode_NotEnabled** (p. 1121) if called on a disabled **DDS::DataWriter** (p. 499):

- The base-class operations **DDS::DataWriter::set_qos** (p. 513), **DDS::DataWriter::get_qos** (p. 514), **DDS::DataWriter::set_listener** (p. 515), **DDS::DataWriter::get_listener** (p. 515),

DDS::Entity::enable (p. 848), **DDS::Entity::get_statuscondition** (p. 849) and **DDS::Entity::get_status_changes** (p. 850)

ˆ **DDS::DataWriter::get_liveliness_lost_status** (p. 504)
DDS::DataWriter::get_offered_deadline_missed_status (p. 504)
DDS::DataWriter::get_offered_incompatible_qos_status (p. 505)
DDS::DataWriter::get_publication_matched_status (p. 505)
DDS::DataWriter::get_reliable_writer_cache_changed_status (p. 505) **DDS::DataWriter::get_reliable_reader_activity_changed_status** (p. 506)

Several **DDS::DataWriter** (p. 499) may operate in different threads. If they share the same **DDS::Publisher** (p. 1044), the middleware guarantees that its operations are thread-safe.

See also:

DDS::TypedDataWriter (p. 1368)
Operations Allowed in Listener Callbacks (p. 954)

Examples:

HelloWorld_publisher.cpp, and **HelloWorldSupport.cpp**.

6.32.2 Member Function Documentation

6.32.2.1 void DDS::DataWriter::get_liveliness_lost_status (LivelinessLostStatus% *status*)

Accesses the DDS::StatusKind::LIVELINESS_LOST_STATUS communication status.

Parameters:

status <<*inout*>> *(p. 176)* **DDS::LivelinessLostStatus** (p. 958) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.32.2.2 void DDS::DataWriter::get_offered_deadline_missed_status (OfferedDeadlineMissedStatus% *status*)

Accesses the DDS::StatusKind::OFFERED_DEADLINE_MISSED_STATUS communication status.

Parameters:

status *<<inout>>* *(p. 176)* **DDS::OfferedDeadlineMissedStatus** (p. 989) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.32.2.3 void DDS::DataWriter::get_offered_incompatible_qos_status (OfferedIncompatibleQosStatus^ *status*)

Accesses the DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS communication status.

Parameters:

status *<<inout>>* *(p. 176)* **DDS::OfferedIncompatibleQosStatus** (p. 991) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.32.2.4 void DDS::DataWriter::get_publication_matched_status (PublicationMatchedStatus% *status*)

Accesses the DDS::StatusKind::PUBLICATION_MATCHED_STATUS communication status.

Parameters:

status *<<inout>>* *(p. 176)* **DDS::PublicationMatchedStatus** (p. 1041) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.32.2.5 void DDS::DataWriter::get_reliable_writer_cache_changed_status (ReliableWriterCacheChangedStatus% *status*)

*<<**eXtension**>>* *(p. 174)* Get the reliable cache status for this writer.

Parameters:

status *<<inout>>* *(p. 176)* **DDS::ReliableWriterCacheChangedStatus** (p. 1101) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.32.2.6 void DDS::DataWriter::get_reliable_reader_activity_changed_status (ReliableReaderActivityChangedStatus% *status*)

<<eXtension>> *(p. 174)* Get the reliable reader activity changed status for this writer.

Parameters:

status *<<inout>>* *(p. 176)* **DDS::ReliableReaderActivityChangedStatus** (p. 1098) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.32.2.7 virtual void DDS::DataWriter::get_datawriter_cache_status (DataWriterCacheStatus% *status*) [virtual]

<<eXtension>> *(p. 174)* Get the datawriter cache status for this writer.

Parameters:

status *<<inout>>* *(p. 176)* **DDS::DataWriterCacheStatus** (p. 523) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_NotEnabled** (p. 1121).

6.32.2.8 virtual void DDS::DataWriter::get_datawriter_protocol_status (DataWriterProtocolStatus% *status*) [virtual]

<<eXtension>> *(p. 174)* Get the datawriter protocol status for this writer.

Parameters:

status *<<inout>>* *(p. 176)* **DDS::DataWriterProtocolStatus** (p. 534) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_NotEnabled** (p. 1121).

6.32.2.9 virtual void DDS::DataWriter::get_matched_subscription_datawriter_protocol_status (DataWriterProtocolStatus% *status*, InstanceHandle_t% *subscription_handle*) [virtual]

*<<**eXtension**>>* *(p. 174)* Get the datawriter protocol status for this writer, per matched subscription identified by the subscription_handle.

Note: Status for a remote entity is only kept while the entity is alive. Once a remote entity is no longer alive, its status is deleted.

Parameters:

status *<<inout>>* *(p. 176)* **DDS::DataWriterProtocolStatus** (p. 534) to be filled in. Cannot be NULL.

subscription_handle *<<in>>* *(p. 175)* Handle to a specific subscription associated with the **DDS::DataReader** (p. 433). Cannot be NULL.Must correspond to a subscription currently associated with the **DDS::DataWriter** (p. 499).

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_NotEnabled** (p. 1121).

6.32.2.10 virtual void DDS::DataWriter::get_matched_subscription_datawriter_protocol_status_by_locator (DataWriterProtocolStatus% *status*, Locator_t^ *locator*) [virtual]

*<<**eXtension**>>* *(p. 174)* Get the datawriter protocol status for this writer, per matched subscription identified by the locator.

Note: Status for a remote entity is only kept while the entity is alive. Once a remote entity is no longer alive, its status is deleted.

Parameters:

status *<<inout>>* *(p. 176)* **DDS::DataWriterProtocolStatus** (p. 534) to be filled in Cannot be NULL.

locator *<<in>>* *(p. 175)* Locator to a specific locator associated with the **DDS::DataReader** (p. 433). Cannot be NULL.Must correspond to a locator of one or more subscriptions currently associated with the **DDS::DataWriter** (p. 499).

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_NotEnabled** (p. 1121).

6.32.2.11 void DDS::DataWriter::assert_liveliness ()

This operation manually asserts the liveliness of this **DDS::DataWriter** (p. 499).

This is used in combination with the **LIVELINESS** (p. 286) policy to indicate to RTI Data Distribution Service that the **DDS::DataWriter** (p. 499) remains active.

You only need to use this operation if the **LIVELINESS** (p. 286) setting is either DDS::LivelinessQosPolicyKind::MANUAL_BY_PARTICIPANT_LIVELINESS_QOS or DDS::LivelinessQosPolicyKind::MANUAL_BY_TOPIC_LIVELINESS_QOS. Otherwise, it has no effect.

Note: writing data via the **DDS::TypedDataWriter::write** (p. 1376) or **DDS::TypedDataWriter::write_w_timestamp** (p. 1378) operation asserts liveliness on the **DDS::DataWriter** (p. 499) itself, and its **DDS::DomainParticipant** (p. 577). Consequently the use of **assert_liveliness()** (p. 508) is only needed if the application is not writing data regularly.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_NotEnabled** (p. 1121)

See also:

DDS::LivelinessQosPolicy (p. 960)

6.32.2.12 virtual void DDS::DataWriter::get_matched_subscription_locators (LocatorSeq^ *locators*) [virtual]

<<eXtension>> (p. 174) Retrieve the list of locators for subscriptions currently "associated" with this **DDS::DataWriter** (p. 499).

Matched subscription locators include locators for all those subscriptions in the same domain that have a matching **DDS::Topic** (p. 1258), compatible QoS and common partition that the **DDS::DomainParticipant** (p. 577) has not indicated should be "ignored" by means of the **DDS::DomainParticipant::ignore_subscription** (p. 636) operation.

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. 433) entities.

Parameters:

locators *<<inout>> (p. 176).* Handles of all the matched subscription locators.

The sequence will be grown if the sequence has ownership and the system has the corresponding resources. Use a sequence without ownership to avoid dynamic memory allocation. If the sequence is too small to store all the matches and the system can not resize the sequence, this method will fail with **DDS::Retcode_OutOfResources** (p. 1122). Cannot be NULL..

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_OutOfResources** (p. 1122) if the sequence is too small and the system can not resize it, or **DDS::Retcode_NotEnabled** (p. 1121)

6.32.2.13 void DDS::DataWriter::get_matched_subscriptions (InstanceHandleSeq^ *subscription_handles*)

Retrieve the list of subscriptions currently "associated" with this **DDS::DataWriter** (p. 499).

Matched subscriptions include all those in the same domain that have a matching **DDS::Topic** (p. 1258), compatible QoS and common partition that the **DDS::DomainParticipant** (p. 577) has not indicated should be "ignored" by means of the **DDS::DomainParticipant::ignore_subscription** (p. 636) operation.

The handles returned in the `subscription_handles` list are the ones that are used by the DDS implementation to locally identify the corresponding matched **DDS::DataReader** (p. 433) entities. These handles match the ones that appear in the **DDS::SampleInfo::instance_handle** (p. 1153) field of the **DDS::SampleInfo** (p. 1148) when reading the **DDS::SubscriptionBuiltinTopicDataTypeSupport::SUBSCRIPTION_TOPIC_NAME** (p. 232) builtin topic.

Parameters:

subscription_handles *<<inout>> (p. 176).* 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 the matches and the system can not resize the sequence, this method will fail with **DDS::Retcode_OutOfResources** (p. 1122).

The maximum number of matches possible is configured with **DDS::DomainParticipantResourceLimitsQosPolicy** (p. 688). You can use a zero-maximum sequence without ownership to quickly check whether there are any matches without allocating any memory. Cannot be NULL..

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_OutOfResources** (p. 1122) if the sequence is too small and the system can not resize it, or **DDS::Retcode_NotEnabled** (p. 1121)

6.32.2.14 void DDS::DataWriter::get_matched_subscription_data (SubscriptionBuiltinTopicData^ *subscription_data*, InstanceHandle_t% *subscription_handle*)

This operation retrieves the information on a subscription that is currently "associated" with the **DDS::DataWriter** (p. 499).

The `subscription_handle` must correspond to a subscription currently associated with the **DDS::DataWriter** (p. 499). Otherwise, the operation will fail and fail with **DDS::Retcode_BadParameter** (p. 1115). Use **DDS::DataWriter::get_matched_subscriptions** (p. 509) to find the subscriptions that are currently matched with the **DDS::DataWriter** (p. 499).

Note: This operation does not retrieve the following information in **DDS::SubscriptionBuiltinTopicData** (p. 1233):

- ^ **DDS::SubscriptionBuiltinTopicData::type_code** (p. 1239)

- DDS::SubscriptionBuiltinTopicData::property
- **DDS::SubscriptionBuiltinTopicData::content_filter_property** (p. 1239)

The above information is available through **DDS::DataReaderListener::on_data_available()** (p. 463) (if a reader listener is installed on the **DDS::SubscriptionBuiltinTopicDataDataReader** (p. 1241)).

Parameters:

subscription_data *<<inout>> (p. 176).* The information to be filled in on the associated subscription. Cannot be NULL.

subscription_handle *<<in>> (p. 175).* Handle to a specific subscription associated with the **DDS::DataReader** (p. 433). Cannot be NULL.. Must correspond to a subscription currently associated with the **DDS::DataWriter** (p. 499).

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_NotEnabled** (p. 1121)

6.32.2.15 Topic ^ DDS::DataWriter::get_topic ()

This operation returns the **DDS::Topic** (p. 1258) associated with the **DDS::DataWriter** (p. 499).

This is the same **DDS::Topic** (p. 1258) that was used to create the **DDS::DataWriter** (p. 499).

Returns:

DDS::Topic (p. 1258) that was used to create the **DDS::DataWriter** (p. 499).

6.32.2.16 Publisher ^ DDS::DataWriter::get_publisher ()

This operation returns the **DDS::Publisher** (p. 1044) to which the **DDS::DataWriter** (p. 499) belongs.

Returns:

DDS::Publisher (p. 1044) to which the **DDS::DataWriter** (p. 499) belongs.

6.32.2.17 void DDS::DataWriter::wait_for_acknowledgments (Duration_t *max_wait*)

Blocks the calling thread until all data written by reliable **DDS::DataWriter** (p. 499) entity is acknowledged, or until timeout expires.

This operation blocks the calling thread until either all data written by the reliable **DDS::DataWriter** (p. 499) entity is acknowledged by all matched reliable **DDS::DataReader** (p. 433) entities, or else 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; a time out indicates that max_wait elapsed before all the data was acknowledged.

If the **DDS::DataWriter** (p. 499) does not have **DDS::ReliabilityQosPolicy** (p. 1094) kind set to RELIABLE the operation will complete immediately with RETCODE_OK.

Parameters:

max_wait *<<in>> (p. 175)* Specifies maximum time to wait for acknowledgements **DDS::Duration_t** (p. 717) .

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_NotEnabled** (p. 1121), **DDS::Retcode_Timeout** (p. 1124)

6.32.2.18 void DDS::DataWriter::wait_for_asynchronous_publishing (Duration_t *max_wait*)

<<eXtension>> (p. 174) 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. 499) is sent and acknowledged (if reliable) by all matched **DDS::DataReader** (p. 433) 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. 433) is complete in addition to what **DDS::DataWriter::wait_for_acknowledgments** (p. 512) provides.

If the **DDS::DataWriter** (p. 499) does not have **DDS::PublishModeQosPolicy** (p. 1077) kind set to

DDS::PublishModeQosPolicyKind::ASYNCHRONOUS_PUBLISH_-MODE_QOS the operation will complete immediately with DDS::Exception::RETCODE_OK.

Parameters:

max_wait *<<in>>* *(p. 175)* Specifies maximum time to wait for acknowledgements **DDS::Duration_t** (p. 717) .

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_-NotEnabled** (p. 1121), **DDS::Retcode_Timeout** (p. 1124)

6.32.2.19 void DDS::DataWriter::set_qos (DataWriterQos^ *qos*)

Sets the writer QoS.

This operation modifies the QoS of the **DDS::DataWriter** (p. 499).

The **DDS::DataWriterQos::user_data** (p. 550), **DDS::DataWriterQos::deadline** (p. 549), **DDS::DataWriterQos::latency_-budget** (p. 549), **DDS::DataWriterQos::ownership_-strength** (p. 550), **DDS::DataWriterQos::transport_-priority** (p. 549), **DDS::DataWriterQos::lifespan** (p. 549) and **DDS::DataWriterQos::writer_data_lifecycle** (p. 550) can be changed. The other policies are immutable.

Parameters:

qos *<<in>>* *(p. 175)* The **DDS::DataWriterQos** (p. 546) to be set to. Policies must be consistent. Immutable policies cannot be changed after **DDS::DataWriter** (p. 499) is enabled. The special value **DDS::Publisher::DATAWRITER_-QOS_DEFAULT** (p. 80) can be used to indicate that the QoS of the **DDS::DataWriter** (p. 499) should be changed to match the current default **DDS::DataWriterQos** (p. 546) set in the **DDS::Publisher** (p. 1044). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_-ImmutablePolicy** (p. 1118) or **DDS::Retcode_-InconsistentPolicy** (p. 1119)

See also:

DDS::DataWriterQos (p. 546) for rules on consistency among QoS

set_qos (abstract) (p. 846)
Operations Allowed in Listener Callbacks (p. 954)

6.32.2.20 void DDS::DataWriter::set_qos_with_profile (System::String^ *library_name*, System::String^ *profile_name*)

<<eXtension>> *(p. 174)* Change the QoS of this writer using the input XML QoS profile.

This operation modifies the QoS of the **DDS::DataWriter** (p. 499).

The **DDS::DataWriterQos::user_data** (p. 550), **DDS::DataWriterQos::deadline** (p. 549), **DDS::DataWriterQos::latency_budget** (p. 549), **DDS::DataWriterQos::ownership_strength** (p. 550), **DDS::DataWriterQos::transport_priority** (p. 549), **DDS::DataWriterQos::lifespan** (p. 549) and **DDS::DataWriterQos::writer_data_lifecycle** (p. 550) can be changed. The other policies are immutable.

Parameters:

library_name *<<in>>* *(p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::Publisher::set_default_library** (p. 1051)).

profile_name *<<in>>* *(p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::Publisher::set_default_profile** (p. 1052)).

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_ImmutablePolicy** (p. 1118) or **DDS::Retcode_InconsistentPolicy** (p. 1119)

See also:

DDS::DataWriterQos (p. 546) for rules on consistency among QoS
Operations Allowed in Listener Callbacks (p. 954)

6.32.2.21 void DDS::DataWriter::get_qos (DataWriterQos^ *qos*)

Gets the writer QoS.

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

Parameters:

qos *<<inout>>* *(p. 176)* The **DDS::DataWriterQos** (p. 546) to be filled up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

get_qos (abstract) (p. 847)

6.32.2.22 void DDS::DataWriter::set_listener (DataWriterListener^ *l*, StatusMask *mask*)

Sets the writer listener.

Parameters:

l *<<in>>* *(p. 175)* **DDS::DataWriterListener** (p. 524) to set to

mask *<<in>>* *(p. 175)* DDS::StatusMask associated with the **DDS::DataWriterListener** (p. 524).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

set_listener (abstract) (p. 847)

6.32.2.23 DataWriterListener ^ DDS::DataWriter::get_listener ()

Get the writer listener.

Returns:

DDS::DataWriterListener (p. 524) of the **DDS::DataWriter** (p. 499).

See also:

get_listener (abstract) (p. 848)

6.32.2.24 void DDS::DataWriter::flush ()

<<eXtension>> (p. 174) 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. 499) does not have **DDS::PublishModeQosPolicy** (p. 1077) kind set to DDS::PublishModeQosPolicyKind::ASYNCHRONOUS_PUBLISH_MODE_QOS, the batch will be sent on the network immediately (in the context of the calling thread).

If the **DDS::DataWriter** (p. 499) does have **DDS::PublishModeQosPolicy** (p. 1077) kind set to DDS::PublishModeQosPolicyKind::ASYNCHRONOUS_PUBLISH_MODE_QOS, the batch will be sent in the context of the asynchronous publishing thread.

This operation may block in the same conditions than **DDS::TypedDataWriter::write** (p. 1376).

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.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_Timeout** (p. 1124), **DDS::Retcode_OutOfResources** (p. 1122) or **DDS::Retcode_NotEnabled** (p. 1121).

6.32.2.25 InstanceHandle_t DDS::DataWriter::register_instance_untyped (System::Object^ *instance_data*)

Register a new instance with this writer.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataWriter** (p. 1368) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataWriter::register_instance** (p. 1370) method instead of this one. See that method for detailed documentation.

See also:

DDS::DataWriter::unregister_instance_untyped (p. 517)
DDS::TypedDataWriter::register_instance (p. 1370)

6.32.2.26 InstanceHandle_t DDS::DataWriter::register_instance_w_timestamp_untyped (System::Object^ *instance_data*, DDS::Time_t% *source_timestamp*)

Register a new instance with this writer using the given time instead of the current time.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataWriter** (p. 1368) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataWriter::register_instance_w_timestamp** (p. 1371) method instead of this one. See that method for detailed documentation.

See also:

DDS::DataWriter::unregister_instance_w_timestamp_untyped (p. 517)
DDS::TypedDataWriter::register_instance (p. 1370)

6.32.2.27 void DDS::DataWriter::unregister_instance_untyped (System::Object^ *instance_data*, DDS::InstanceHandle_t% *handle*)

Unregister a new instance from this writer.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataWriter** (p. 1368) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataWriter::unregister_instance** (p. 1372) method instead of this one. See that method for detailed documentation.

See also:

DDS::DataWriter::register_instance_untyped (p. 516)
DDS::TypedDataWriter::unregister_instance (p. 1372)

6.32.2.28 void DDS::DataWriter::unregister_instance_w_timestamp_untyped (System::Object^ *instance_data*, DDS::InstanceHandle_t% *handle*, DDS::Time_t% *source_timestamp*)

Unregister a new instance from this writer using the given time instead of the current time.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataWriter** (p. 1368) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataWriter::unregister_instance_w_timestamp** (p. 1374) method instead of this one. See that method for detailed documentation.

See also:

DDS::DataWriter::register_instance_w_timestamp_untyped (p. 517)
DDS::TypedDataWriter::unregister_instance_w_timestamp (p. 1374)

6.32.2.29 void DDS::DataWriter::write_untyped (System::Object^ *instance_data*, DDS::InstanceHandle_t% *handle*)

Publish a data sample.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataWriter** (p. 1368) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataWriter::write** (p. 1376) method instead of this one. See that method for detailed documentation.

See also:

DDS::DataWriter::write_w_timestamp_untyped (p. 518)
DDS::TypedDataWriter::write (p. 1376)

6.32.2.30 void DDS::DataWriter::write_w_timestamp_untyped (System::Object^ *instance_data*, DDS::InstanceHandle_t% *handle*, DDS::Time_t% *source_timestamp*)

Publish a data sample using the given time instead of the current time.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataWriter** (p. 1368) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataWriter::write_w_timestamp** (p. 1378) method instead of this one. See that method for detailed documentation.

See also:

DDS::DataWriter::write_untyped (p. 518)
DDS::TypedDataWriter::write_w_timestamp (p. 1378)

6.32.2.31 void DDS::DataWriter::dispose_untyped (System::Object^ *instance_data*, DDS::InstanceHandle_t% *instance_handle*)

Dispose a data sample.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataWriter** (p. 1368) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataWriter::dispose** (p. 1379) method instead of this one. See that method for detailed documentation.

See also:

DDS::DataWriter::dispose_w_timestamp_untyped (p. 519)
DDS::TypedDataWriter::dispose (p. 1379)

6.32.2.32 void DDS::DataWriter::dispose_w_timestamp_untyped (System::Object^ *instance_data*, DDS::InstanceHandle_t% *instance_handle*, DDS::Time_t% *source_timestamp*)

Dispose a data sample using the given time instead of the current time.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataWriter** (p. 1368) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataWriter::dispose_w_timestamp** (p. 1381) method instead of this one. See that method for detailed documentation.

See also:

DDS::DataWriter::dispose_untyped (p. 519)
DDS::TypedDataWriter::dispose_w_timestamp (p. 1381)

6.32.2.33 void DDS::DataWriter::get_key_value_untyped (System::Object^ *key_holder*, DDS::InstanceHandle_t% *handle*)

Fill in the key fields of the given data sample.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataWriter** (p. 1368) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataWriter::get_key_value** (p. 1383) method instead of this one. See that method for detailed documentation.

See also:

DDS::TypedDataWriter::get_key_value (p. 1383)

6.32.2.34 InstanceHandle_t DDS::DataWriter::lookup_instance_untyped (System::Object^ *key_holder*)

Given a sample with the given key field values, return the handle corresponding to its instance.

This method allows type-indepedent code to work with a variety of concrete **DDS::TypedDataWriter** (p. 1368) classes in a consistent way.

Statically type-safe code should use the appropriate **DDS::TypedDataWriter::lookup_instance** (p. 1384) method instead of this one. See that method for detailed documentation.

See also:

DDS::TypedDataWriter::lookup_instance (p. 1384)

6.32.2.35 virtual void DDS::DataWriter::enable () `[override, virtual]`

Enables the **DDS::Entity** (p. 845).

This operation enables the **Entity** (p. 845). **Entity** (p. 845) objects can be created either enabled or disabled. This is controlled by the value of the **ENTITY_FACTORY** (p. 304) QoS policy on the corresponding factory for the **DDS::Entity** (p. 845).

By default, **ENTITY_FACTORY** (p. 304) is set so that it is not necessary to explicitly call **DDS::Entity::enable** (p. 848) on newly created entities.

The **DDS::Entity::enable** (p. 848) operation is idempotent. Calling enable on an already enabled **Entity** (p. 845) returns OK and has no effect.

If a **DDS::Entity** (p. 845) 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. 849)
- 'factory' operations
- **DDS::Entity::get_status_changes** (p. 850) 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_NotEnabled** (p. 1121).

It is legal to delete an **DDS::Entity** (p. 845) that has not been enabled by calling the proper operation on its factory.

Entities created from a factory that is disabled are created disabled, regardless of the setting of the **DDS::EntityFactoryQosPolicy** (p. 851).

Calling enable on an **Entity** (p. 845) whose factory is not enabled will fail and return **DDS::Retcode_PreconditionNotMet** (p. 1123).

If **DDS::EntityFactoryQosPolicy::autoenable_created_entities** (p. 852) is TRUE, the enable operation on a factory will automatically enable all entities created from that factory.

Listeners associated with an entity are not called until the entity is enabled.

Conditions associated with a disabled entity are "inactive," that is, they have a `trigger_value` == FALSE.

Exceptions:

One of the **Standard Return Codes** (p. 235), **Standard Return Codes** (p. 235) or **DDS::Retcode_PreconditionNotMet** (p. 1123).

Implements **DDS::Entity** (p. 848).

6.32.2.36 virtual StatusCondition ^ DDS::DataWriter::get_statuscondition () `[override, virtual]`

Allows access to the **DDS::StatusCondition** (p. 1183) associated with the **DDS::Entity** (p. 845).

The returned condition can then be added to a **DDS::WaitSet** (p. 1411) so that the application can wait for specific status changes that affect the **DDS::Entity** (p. 845).

Returns:

the status condition associated with this entity.

Implements **DDS::Entity** (p. 849).

6.32.2.37 virtual StatusMask DDS::DataWriter::get_status_changes () [override, virtual]

Retrieves the list of communication statuses in the **DDS::Entity** (p. 845) that are triggered.

That is, the list of statuses whose value has changed since the last time the application read the status using the get_*_status() method.

When the entity is first created or if the entity is not enabled, all communication statuses are in the "untriggered" state so the list returned by the get_status_changes operation will be empty.

The list of statuses returned by the get_status_changes operation refers to the status that are triggered on the **Entity** (p. 845) itself and does not include statuses that apply to contained entities.

Returns:

list of communication statuses in the **DDS::Entity** (p. 845) that are triggered.

See also:

Status Kinds (p. 238)

Implements **DDS::Entity** (p. 850).

6.32.2.38 virtual InstanceHandle_t DDS::DataWriter::get_instance_handle () [override, virtual]

Allows access to the **DDS::InstanceHandle_t** (p. 905) associated with the **DDS::Entity** (p. 845).

This operation returns the **DDS::InstanceHandle_t** (p. 905) that represents the **DDS::Entity** (p. 845).

Returns:

the instance handle associated with this entity.

Implements **DDS::Entity** (p. 850).

6.33 DDS::DataWriterCacheStatus Struct Reference

<<*eXtension*>> *(p. 174)* The status of the writer's cache.

`#include <managed_publication.h>`

Public Attributes

- System::Int64 **sample_count_peak**

  Highest number of samples in the writer's queue over the lifetime of the writer.

- System::Int64 **sample_count**

  Number of samples in the writer's queue.

6.33.1 Detailed Description

<<*eXtension*>> *(p. 174)* The status of the writer's cache.

Entity:

DDS::DataWriter (p. 499)

6.33.2 Member Data Documentation

6.33.2.1 System::Int64 DDS::DataWriterCacheStatus::sample_count_peak

Highest number of samples in the writer's queue over the lifetime of the writer.

6.33.2.2 System::Int64 DDS::DataWriterCacheStatus::sample_count

Number of samples in the writer's queue.

6.34 DDS::DataWriterListener Class Reference

<<*interface*>> *(p. 175)* **DDS::Listener** (p. 952) for writer status.

`#include <managed_publication.h>`

Inheritance diagram for DDS::DataWriterListener::

Public Member Functions

- ^ virtual void **on_offered_deadline_missed** (**DataWriter**^ writer, **OfferedDeadlineMissedStatus**% status)

  Handles the DDS::StatusKind::OFFERED_DEADLINE_MISSED_STATUS status.

- ^ virtual void **on_liveliness_lost** (**DataWriter**^ writer, **LivelinessLostStatus**% status)

  Handles the DDS::StatusKind::LIVELINESS_LOST_STATUS status.

- ^ virtual void **on_offered_incompatible_qos** (**DataWriter**^ writer, **OfferedIncompatibleQosStatus**^ status)

  Handles the DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS status.

- ^ virtual void **on_publication_matched** (**DataWriter**^ writer, **PublicationMatchedStatus**% status)

  Handles the DDS::StatusKind::PUBLICATION_MATCHED_STATUS status.

- ^ virtual void **on_reliable_writer_cache_changed** (**DataWriter**^ writer, **ReliableWriterCacheChangedStatus**% status)

  <<**eXtension**>> (p. 174) *A change has occurred in the writer's cache of unacknowledged samples.*

- ^ virtual void **on_reliable_reader_activity_changed** (**DataWriter**^ writer, **ReliableReaderActivityChangedStatus**% status)

  <<**eXtension**>> (p. 174) *A matched reliable reader has become active or become inactive.*

- ^ virtual void **on_instance_replaced** (**DataWriter**^ writer, **InstanceHandle_t**% handle)

  *Notifies when an instance is replaced in **DataWriter** (p. 499) queue.*

6.34.1 Detailed Description

<<*interface*>> *(p. 175)* **DDS::Listener** (p. 952) for writer status.

Entity:

DDS::DataWriter (p. 499)

Status:

DDS::StatusKind::LIVELINESS_LOST_STATUS,
DDS::LivelinessLostStatus (p. 958);
DDS::StatusKind::OFFERED_DEADLINE_MISSED_STATUS,
DDS::OfferedDeadlineMissedStatus (p. 989);
DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS,
DDS::OfferedIncompatibleQosStatus (p. 991);
DDS::StatusKind::PUBLICATION_MATCHED_STATUS,
DDS::PublicationMatchedStatus (p. 1041);
DDS::StatusKind::RELIABLE_READER_ACTIVITY_CHANGED_STATUS, **DDS::ReliableReaderActivityChangedStatus** (p. 1098);
DDS::StatusKind::RELIABLE_WRITER_CACHE_CHANGED_STATUS,
DDS::ReliableWriterCacheChangedStatus (p. 1101);

See also:

Status Kinds (p. 238)
Operations Allowed in Listener Callbacks (p. 954)

6.34.2 Member Function Documentation

6.34.2.1 virtual void DDS::DataWriterListener::on_offered_deadline_missed (DataWriter^ *writer*, OfferedDeadlineMissedStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::OFFERED_DEADLINE_MISSED_STATUS status.

This callback is called when the deadline that the **DDS::DataWriter** (p. 499) has committed through its **DEADLINE** (p. 281) qos policy was not respected for a specific instance. This callback is called for each deadline period elapsed during which the **DDS::DataWriter** (p. 499) failed to provide data for an instance.

Parameters:

writer <<*out*>> *(p. 176)* Locally created **DDS::DataWriter** (p. 499) that triggers the listener callback

status <<***out***>> *(p. 176)* Current deadline missed status of locally created **DDS::DataWriter** (p. 499)

Reimplemented in **DDS::DomainParticipantListener** (p. 678), and **DDS::PublisherListener** (p. 1070).

6.34.2.2 virtual void DDS::DataWriterListener::on_liveliness_lost (DataWriter^ *writer*, LivelinessLostStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::LIVELINESS_LOST_STATUS status.

This callback is called when the liveliness that the **DDS::DataWriter** (p. 499) has committed through its **LIVELINESS** (p. 286) qos policy was not respected; this **DDS::DataReader** (p. 433) entities will consider the **DDS::DataWriter** (p. 499) as no longer "alive/active". This callback will not be called when an already not alive **DDS::DataWriter** (p. 499) simply renames not alive for another liveliness period.

Parameters:

writer <<***out***>> *(p. 176)* Locally created **DDS::DataWriter** (p. 499) that triggers the listener callback

status <<***out***>> *(p. 176)* Current liveliness lost status of locally created **DDS::DataWriter** (p. 499)

Reimplemented in **DDS::DomainParticipantListener** (p. 679), and **DDS::PublisherListener** (p. 1071).

6.34.2.3 virtual void DDS::DataWriterListener::on_offered_incompatible_qos (DataWriter^ *writer*, OfferedIncompatibleQosStatus^ *status*) [inline, virtual]

Handles the DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS status.

This callback is called when the **DDS::DataWriterQos** (p. 546) of the **DDS::DataWriter** (p. 499) was incompatible with what was requested by a **DDS::DataReader** (p. 433). This callback is called when a **DDS::DataWriter** (p. 499) has discovered a **DDS::DataReader** (p. 433) for the same **DDS::Topic** (p. 1258) and common partition, but with a requested QoS that is incompatible with that offered by the **DDS::DataWriter** (p. 499).

Parameters:

writer <<***out***>> *(p. 176)* Locally created **DDS::DataWriter** (p. 499) that triggers the listener callback

status *<<**out**>>* *(p. 176)* Current incompatible qos status of locally created **DDS::DataWriter** (p. 499)

Reimplemented in **DDS::DomainParticipantListener** (p. 679), and **DDS::PublisherListener** (p. 1071).

6.34.2.4 virtual void DDS::DataWriterListener::on_publication_matched (DataWriter^ *writer*, PublicationMatchedStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::PUBLICATION_MATCHED_STATUS status.

This callback is called when the **DDS::DataWriter** (p. 499) has found a **DDS::DataReader** (p. 433) that matches the **DDS::Topic** (p. 1258), has a common partition and compatible QoS, or has ceased to be matched with a **DDS::DataReader** (p. 433) that was previously considered to be matched.

Parameters:

writer *<<**out**>>* *(p. 176)* Locally created **DDS::DataWriter** (p. 499) that triggers the listener callback

status *<<**out**>>* *(p. 176)* Current publication match status of locally created **DDS::DataWriter** (p. 499)

Reimplemented in **DDS::DomainParticipantListener** (p. 680), and **DDS::PublisherListener** (p. 1072).

6.34.2.5 virtual void DDS::DataWriterListener::on_reliable_writer_cache_changed (DataWriter^ *writer*, ReliableWriterCacheChangedStatus% *status*) [inline, virtual]

*<<**eXtension**>>* *(p. 174)* A change has occurred in the writer's cache of unacknowledged samples.

Parameters:

writer *<<**out**>>* *(p. 176)* Locally created **DDS::DataWriter** (p. 499) that triggers the listener callback

status *<<**out**>>* *(p. 176)* Current reliable writer cache changed status of locally created **DDS::DataWriter** (p. 499)

Reimplemented in **DDS::DomainParticipantListener** (p. 680), and **DDS::PublisherListener** (p. 1072).

6.34.2.6 virtual void DDS::DataWriterListener::on_reliable_reader_activity_changed (DataWriter^ *writer*, ReliableReaderActivityChangedStatus% *status*) [inline, virtual]

<<eXtension>> (p. 174) A matched reliable reader has become active or become inactive.

Parameters:

writer *<<out>> (p. 176)* Locally created **DDS::DataWriter** (p. 499) that triggers the listener callback

status *<<out>> (p. 176)* Current reliable reader activity changed status of locally created **DDS::DataWriter** (p. 499)

Reimplemented in **DDS::DomainParticipantListener** (p. 680), and **DDS::PublisherListener** (p. 1073).

6.34.2.7 virtual void DDS::DataWriterListener::on_instance_replaced (DataWriter^ *writer*, InstanceHandle_t% *handle*) [inline, virtual]

Notifies when an instance is replaced in **DataWriter** (p. 499) queue.

This callback is called when an instance is replaced by the **DDS::DataWriter** (p. 499) 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.

Parameters:

writer *<<out>> (p. 176)* Locally created **DDS::DataWriter** (p. 499) that triggers the listener callback

handle *<<out>> (p. 176)* Handle of the replaced instance

Reimplemented in **DDS::PublisherListener** (p. 1073).

6.35 DDS::DataWriterProtocolQosPolicy Struct Reference

Protocol that applies only to **DDS::DataWriter** (p. 499) instances.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_datawriterprotocol_qos_policy_name** ()

  *Stringified human-readable name for **DDS::DataWriterProtocolQosPolicy** (p. 529).*

Public Attributes

- **GUID_t virtual_guid**

  The virtual GUID (Global Unique Identifier).

- System::UInt32 **rtps_object_id**

  The RTPS Object ID.

- **RtpsReliableWriterProtocol_t rtps_reliable_writer**

  The reliable protocol defined in RTPS.

Properties

- System::Boolean **push_on_write** `[get, set]`

  Whether to push sample out when write is called.

- System::Boolean **disable_positive_acks** `[get, set]`

  Controls whether or not the writer expects positive acknowledgements from matching readers.

- System::Boolean **disable_inline_keyhash** `[get, set]`

  Controls whether or not a keyhash is propagated on the wire with each sample.

- System::Boolean **serialize_key_with_dispose** `[get, set]`

  Controls whether or not the serialized key is propagated on the wire with dispose samples.

6.35.1 Detailed Description

Protocol that applies only to **DDS::DataWriter** (p. 499) 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. 529) 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** (p. 499) or **DataReader** (p. 433) 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. 499) and **DDS::DataReader** (p. 433) 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 Data Distribution Service responds to "slow" reliable DataReaders or ones that disconnect or are otherwise lost. See **DDS::ReliabilityQosPolicy** (p. 1094) for more information on the per-DataReader/DataWriter reliability configuration. **DDS::HistoryQosPolicy** (p. 898) and **DDS::ResourceLimitsQosPolicy** (p. 1109) also play an important role in the DDS reliable protocol.

This QoS policy is an extension to the DDS standard.

Entity:

DDS::DataWriter (p. 499)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)

6.35.2 Member Data Documentation

6.35.2.1 GUID_t DDS::DataWriterProtocolQosPolicy::virtual_guid

The virtual GUID (Global Unique Identifier).

The virtual GUID is used to uniquely identify different incarnations of the same **DDS::DataWriter** (p. 499).

RTI Data Distribution Service uses the virtual GUID to associate a persisted writer history to a specific **DDS::DataWriter** (p. 499).

The RTI Data Distribution Service Persistence Service uses the virtual GUID to send samples on behalf of the original **DDS::DataWriter** (p. 499).

[**default**] DDS::Guid_t::GUID_AUTO

6.35.2.2 System::UInt32 DDS::DataWriterProtocolQosPolicy::rtps_object_id

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 Data Distribution Service 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 Data Distribution Service. In those cases, the recomendation is not to use automatic object ID assigment.

[**default**] **DDS::WireProtocolQosPolicy::RTPS_AUTO_ID** (p. 330)

[**range**] [0,0x00ffffff]

6.35.2.3 RtpsReliableWriterProtocol_t DDS::DataWriterProtocolQosPolicy::rtps_reliable_writer

The reliable protocol defined in RTPS.

[**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;

max_heartbeat_retries 10;

inactivate_nonprogressing_readers DDS_BOOLEAN_FALSE;

heartbeats_per_max_samples 8;

min_nack_response_delay 0.0 seconds;

max_nack_response_delay 0.2 seconds;

max_bytes_per_nack_response 131072

6.35.3 Property Documentation

6.35.3.1 System:: Boolean DDS::DataWriterProtocolQosPolicy::push_on_write [get, set]

Whether to push sample out when write is called.

If set to true (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.

Note: push_on_write must be TRUE for Asynchronous DataWriters (those with DDS::PublishModeQosPolicyKind::ASYNCHRONOUS_PUBLISH_MODE_QOS). Otherwise, samples will never be sent.

[default] true

6.35.3.2 System:: Boolean DDS::DataWriterProtocolQosPolicy::disable_positive_acks [get, set]

Controls whether or not the writer expects positive acknowledgements from matching readers.

If set to true, the writer does not expect readers to send send positive acknowledgments to the writer. Consequently, instead of keeping a sample queued until all readers have positively acknowledged it, the writer will keep a sample for at least **DDS::RtpsReliableWriterProtocol_t::disable_positive_acks_min_sample_keep_duration** (p. 1136), after which the sample is logically considered as positively acknowledged.

If set to false (the default), the writer expects to receive positive acknowledgements from its acknowledging readers (**DDS::DataReaderProtocolQosPolicy::disable_positive_acks** (p. 468) = false) and it applies the keep-duration to its non-acknowledging readers (**DDS::DataReaderProtocolQosPolicy::disable_positive_acks** (p. 468) = true).

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] false

6.35.3.3 System:: Boolean DDS::DataWriterProtocolQosPolicy::disable_inline_keyhash [get, set]

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 Data Distribution Service associates an internal 16-byte representation, called a keyhash.

When this field is false, the keyhash is sent on the wire with every data instance.

When this field is true, 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 true is not compatible with using RTI Real-Time Connect or RTI Recorder.

[**default**] false

6.35.3.4 System:: Boolean DDS::DataWriterProtocolQosPolicy::serialize_key_with_dispose [get, set]

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 true if there are DataReaders where **DDS::DataReaderProtocolQosPolicy::propagate_dispose_of_unregistered_instances** (p. 469) is also true.

Important: When this field is true, batching will not be compatible with RTI Data Distribution Service 4.3e, 4.4b, or 4.4c. The **DDS::DataReader** (p. 433) entities will receive incorrect data and/or encounter deserialization errors.

[**default**] false

6.36 DDS::DataWriterProtocolStatus Struct Reference

*<<**eXtension**>>* *(p. 174)* The status of a writer's internal protocol related metrics, like the number of samples pushed, pulled, filtered; and status of wire protocol traffic.

`#include <managed_publication.h>`

Public Attributes

- System::Int64 **pushed_sample_count**

  *The number of user samples pushed on write from a local **DataWriter** (p. 499) to a matching remote **DataReader** (p. 433).*

- System::Int64 **pushed_sample_count_change**

  *The incremental change in the number of user samples pushed on write from a local **DataWriter** (p. 499) to a matching remote **DataReader** (p. 433) since the last time the status was read.*

- System::Int64 **pushed_sample_bytes**

  *The number of bytes of user samples pushed on write from a local **DataWriter** (p. 499) to a matching remote **DataReader** (p. 433).*

- System::Int64 **pushed_sample_bytes_change**

  *The incremental change in the number of bytes of user samples pushed on write from a local **DataWriter** (p. 499) to a matching remote **DataReader** (p. 433) since the last time the status was read.*

- System::Int64 **filtered_sample_count**

  *The number of user samples preemptively filtered by a local **DataWriter** (p. 499) due to Content-Filtered Topics.*

- System::Int64 **filtered_sample_count_change**

  *The incremental change in the number of user samples preemptively filtered by a local **DataWriter** (p. 499) due to Content-Filtered Topics since the last time the status was read.*

- System::Int64 **filtered_sample_bytes**

  *The number of user samples preemptively filtered by a local **DataWriter** (p. 499) due to Content-Filtered Topics.*

- System::Int64 **filtered_sample_bytes_change**

*The incremental change in the number of user samples preemptively filtered by a local **DataWriter** (p. 499) due to Content-Filtered Topics since the last time the status was read.*

^ System::Int64 **sent_heartbeat_count**

*The number of Heartbeats sent between a local **DataWriter** (p. 499) and matching remote **DataReader** (p. 433).*

^ System::Int64 **sent_heartbeat_count_change**

*The incremental change in the number of Heartbeats sent between a local **DataWriter** (p. 499) and matching remote **DataReader** (p. 433) since the last time the status was read.*

^ System::Int64 **sent_heartbeat_bytes**

*The number of bytes of Heartbeats sent between a local **DataWriter** (p. 499) and matching remote **DataReader** (p. 433).*

^ System::Int64 **sent_heartbeat_bytes_change**

*The incremental change in the number of bytes of Heartbeats sent between a local **DataWriter** (p. 499) and matching remote **DataReader** (p. 433) since the last time the status was read.*

^ System::Int64 **pulled_sample_count**

*The number of user samples pulled from local **DataWriter** (p. 499) by matching DataReaders.*

^ System::Int64 **pulled_sample_count_change**

*The incremental change in the number of user samples pulled from local **DataWriter** (p. 499) by matching DataReaders since the last time the status was read.*

^ System::Int64 **pulled_sample_bytes**

*The number of bytes of user samples pulled from local **DataWriter** (p. 499) by matching DataReaders.*

^ System::Int64 **pulled_sample_bytes_change**

*The incremental change in the number of bytes of user samples pulled from local **DataWriter** (p. 499) by matching DataReaders since the last time the status was read.*

^ System::Int64 **received_ack_count**

*The number of ACKs from a remote **DataReader** (p. 433) received by a local **DataWriter** (p. 499).*

^ System::Int64 **received_ack_count_change**

*The incremental change in the number of ACKs from a remote **DataReader*** (p. 433) *received by a local **DataWriter*** (p. 499) *since the last time the status was read.*

^ System::Int64 **received_ack_bytes**

*The number of bytes of ACKs from a remote **DataReader*** (p. 433) *received by a local **DataWriter*** (p. 499).

^ System::Int64 **received_ack_bytes_change**

*The incremental change in the number of bytes of ACKs from a remote **DataReader*** (p. 433) *received by a local **DataWriter*** (p. 499) *since the last time the status was read.*

^ System::Int64 **received_nack_count**

*The number of NACKs from a remote **DataReader*** (p. 433) *received by a local **DataWriter*** (p. 499).

^ System::Int64 **received_nack_count_change**

*The incremental change in the number of NACKs from a remote **DataReader*** (p. 433) *received by a local **DataWriter*** (p. 499) *since the last time the status was read.*

^ System::Int64 **received_nack_bytes**

*The number of bytes of NACKs from a remote **DataReader*** (p. 433) *received by a local **DataWriter*** (p. 499).

^ System::Int64 **received_nack_bytes_change**

*The incremental change in the number of bytes of NACKs from a remote **DataReader*** (p. 433) *received by a local **DataWriter*** (p. 499) *since the last time the status was read.*

^ System::Int64 **sent_gap_count**

*The number of GAPs sent from local **DataWriter*** (p. 499) *to matching remote DataReaders.*

^ System::Int64 **sent_gap_count_change**

*The incremental change in the number of GAPs sent from local **DataWriter*** (p. 499) *to matching remote DataReaders since the last time the status was read.*

^ System::Int64 **sent_gap_bytes**

*The number of bytes of GAPs sent from local **DataWriter*** (p. 499) *to matching remote DataReaders.*

^ System::Int64 **sent_gap_bytes_change**

*The incremental change in the number of bytes of GAPs sent from local **DataWriter** (p. 499) to matching remote DataReaders since the last time the status was read.*

ˆ System::Int64 **rejected_sample_count**

The number of times a sample is rejected due to exceptions in the send path.

ˆ System::Int64 **rejected_sample_count_change**

The incremental change in the number of times a sample is rejected due to exceptions in the send path since the last time the status was read.

ˆ System::Int32 **send_window_size**

Current maximum number of outstanding samples allowed in the DataWriter's queue.

ˆ **SequenceNumber_t first_available_sample_sequence_number**

*The sequence number of the first available sample currently queued in the local **DataWriter** (p. 499).*

ˆ **SequenceNumber_t last_available_sample_sequence_number**

*The sequence number of the last available sample currently queued in the local **DataWriter** (p. 499).*

ˆ **SequenceNumber_t first_unacknowledged_sample_sequence_number**

*The sequence number of the first unacknowledged sample currently queued in the local **DataWriter** (p. 499).*

ˆ **SequenceNumber_t first_available_sample_virtual_sequence_number**

*The virtual sequence number of the first available sample currently queued in the local **DataWriter** (p. 499).*

ˆ **SequenceNumber_t last_available_sample_virtual_sequence_number**

*The virtual sequence number of the last available sample currently queued in the local **DataWriter** (p. 499).*

ˆ **SequenceNumber_t first_unacknowledged_sample_virtual_sequence_number**

*The virtual sequence number of the first unacknowledged sample currently queued in the local **DataWriter** (p. 499).*

ˆ **InstanceHandle_t first_unacknowledged_sample_subscription_handle**

*The handle of a remote **DataReader** (p. 433) that has not acknowledged the first unacknowledged sample of the local **DataWriter** (p. 499).*

^ **SequenceNumber_t 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. 532) is set.*

6.36.1 Detailed Description

<<***eXtension***>> *(p. 174)* 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. 499)

6.36.2 Member Data Documentation

6.36.2.1 System::Int64 DDS::DataWriterProtocolStatus::pushed_sample_count

The number of user samples pushed on write from a local **DataWriter** (p. 499) to a matching remote **DataReader** (p. 433).

6.36.2.2 System::Int64 DDS::DataWriterProtocolStatus::pushed_sample_count_change

The incremental change in the number of user samples pushed on write from a local **DataWriter** (p. 499) to a matching remote **DataReader** (p. 433) since the last time the status was read.

6.36.2.3 System::Int64 DDS::DataWriterProtocolStatus::pushed_sample_bytes

The number of bytes of user samples pushed on write from a local **DataWriter** (p. 499) to a matching remote **DataReader** (p. 433).

6.36.2.4 System::Int64 DDS::DataWriterProtocolStatus::pushed_sample_bytes_change

The incremental change in the number of bytes of user samples pushed on write from a local **DataWriter** (p. 499) to a matching remote **DataReader** (p. 433) since the last time the status was read.

6.36.2.5 System::Int64 DDS::DataWriterProtocolStatus::filtered_sample_count

The number of user samples preemptively filtered by a local **DataWriter** (p. 499) due to Content-Filtered Topics.

6.36.2.6 System::Int64 DDS::DataWriterProtocolStatus::filtered_sample_count_change

The incremental change in the number of user samples preemptively filtered by a local **DataWriter** (p. 499) due to Content-Filtered Topics since the last time the status was read.

6.36.2.7 System::Int64 DDS::DataWriterProtocolStatus::filtered_sample_bytes

The number of user samples preemptively filtered by a local **DataWriter** (p. 499) due to Content-Filtered Topics.

6.36.2.8 System::Int64 DDS::DataWriterProtocolStatus::filtered_sample_bytes_change

The incremental change in the number of user samples preemptively filtered by a local **DataWriter** (p. 499) due to Content-Filtered Topics since the last time the status was read.

6.36.2.9 System::Int64 DDS::DataWriterProtocolStatus::sent_heartbeat_count

The number of Heartbeats sent between a local **DataWriter** (p. 499) and matching remote **DataReader** (p. 433).

6.36.2.10 System::Int64 DDS::DataWriterProtocolStatus::sent_heartbeat_count_change

The incremental change in the number of Heartbeats sent between a local **DataWriter** (p. 499) and matching remote **DataReader** (p. 433) since the last time the status was read.

6.36.2.11 System::Int64 DDS::DataWriterProtocolStatus::sent_heartbeat_bytes

The number of bytes of Heartbeats sent between a local **DataWriter** (p. 499) and matching remote **DataReader** (p. 433).

6.36.2.12 System::Int64 DDS::DataWriterProtocolStatus::sent_heartbeat_bytes_change

The incremental change in the number of bytes of Heartbeats sent between a local **DataWriter** (p. 499) and matching remote **DataReader** (p. 433) since the last time the status was read.

6.36.2.13 System::Int64 DDS::DataWriterProtocolStatus::pulled_sample_count

The number of user samples pulled from local **DataWriter** (p. 499) by matching DataReaders.

Pulled samples are samples sent for repairs, for late joiners, and all samples sent by the local **DataWriter** (p. 499) when **DDS::DataWriterProtocolQosPolicy::push_on_write** (p. 532) is false.

6.36.2.14 System::Int64 DDS::DataWriterProtocolStatus::pulled_sample_count_change

The incremental change in the number of user samples pulled from local **DataWriter** (p. 499) by matching DataReaders 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** (p. 499) when **DDS::DataWriterProtocolQosPolicy::push_on_write** (p. 532) is false.

6.36.2.15 System::Int64 DDS::DataWriterProtocolStatus::pulled_sample_bytes

The number of bytes of user samples pulled from local **DataWriter** (p. 499) by matching DataReaders.

Pulled samples are samples sent for repairs, for late joiners, and all samples sent by the local **DataWriter** (p. 499) when **DDS::DataWriterProtocolQosPolicy::push_on_write** (p. 532) is false.

6.36.2.16 System::Int64 DDS::DataWriterProtocolStatus::pulled_sample_bytes_change

The incremental change in the number of bytes of user samples pulled from local **DataWriter** (p. 499) by matching DataReaders 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** (p. 499) when **DDS::DataWriterProtocolQosPolicy::push_on_write** (p. 532) is false.

6.36.2.17 System::Int64 DDS::DataWriterProtocolStatus::received_ack_count

The number of ACKs from a remote **DataReader** (p. 433) received by a local **DataWriter** (p. 499).

6.36.2.18 System::Int64 DDS::DataWriterProtocolStatus::received_ack_count_change

The incremental change in the number of ACKs from a remote **DataReader** (p. 433) received by a local **DataWriter** (p. 499) since the last time the status was read.

6.36.2.19 System::Int64 DDS::DataWriterProtocolStatus::received_ack_bytes

The number of bytes of ACKs from a remote **DataReader** (p. 433) received by a local **DataWriter** (p. 499).

6.36.2.20 System::Int64 DDS::DataWriterProtocolStatus::received_ack_bytes_change

The incremental change in the number of bytes of ACKs from a remote **DataReader** (p. 433) received by a local **DataWriter** (p. 499) since the last time the status was read.

6.36.2.21 System::Int64 DDS::DataWriterProtocolStatus::received_nack_count

The number of NACKs from a remote **DataReader** (p. 433) received by a local **DataWriter** (p. 499).

6.36.2.22 System::Int64 DDS::DataWriterProtocolStatus::received_nack_count_change

The incremental change in the number of NACKs from a remote **DataReader** (p. 433) received by a local **DataWriter** (p. 499) since the last time the status was read.

6.36.2.23 System::Int64 DDS::DataWriterProtocolStatus::received_nack_bytes

The number of bytes of NACKs from a remote **DataReader** (p. 433) received by a local **DataWriter** (p. 499).

6.36.2.24 System::Int64 DDS::DataWriterProtocolStatus::received_nack_bytes_change

The incremental change in the number of bytes of NACKs from a remote **DataReader** (p. 433) received by a local **DataWriter** (p. 499) since the last time the status was read.

6.36.2.25 System::Int64 DDS::DataWriterProtocolStatus::sent_gap_count

The number of GAPs sent from local **DataWriter** (p. 499) to matching remote DataReaders.

6.36.2.26 System::Int64 DDS::DataWriterProtocolStatus::sent_gap_count_change

The incremental change in the number of GAPs sent from local **DataWriter** (p. 499) to matching remote DataReaders since the last time the status was read.

6.36.2.27 System::Int64 DDS::DataWriterProtocolStatus::sent_gap_bytes

The number of bytes of GAPs sent from local **DataWriter** (p. 499) to matching remote DataReaders.

6.36.2.28 System::Int64 DDS::DataWriterProtocolStatus::sent_gap_bytes_change

The incremental change in the number of bytes of GAPs sent from local **DataWriter** (p. 499) to matching remote DataReaders since the last time the status was read.

6.36.2.29 System::Int64 DDS::DataWriterProtocolStatus::rejected_sample_count

The number of times a sample is rejected due to exceptions in the send path.

6.36.2.30 System::Int64 DDS::DataWriterProtocolStatus::rejected_sample_count_change

The incremental change in the number of times a sample is rejected due to exceptions in the send path since the last time the status was read.

6.36.2.31 System::Int32 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. 1137) to **DDS::RtpsReliableWriterProtocol_t::max_send_window_size** (p. 1138).

6.36.2.32 SequenceNumber_t DDS::DataWriterProtocolStatus::first_available_sample_sequence_number

The sequence number of the first available sample currently queued in the local **DataWriter** (p. 499).

6.36.2.33 SequenceNumber_t DDS::DataWriterProtocolStatus::last_available_sample_sequence_number

The sequence number of the last available sample currently queued in the local **DataWriter** (p. 499).

6.36.2.34 SequenceNumber_t DDS::DataWriterProtocolStatus::first_unacknowledged_sample_sequence_number

The sequence number of the first unacknowledged sample currently queued in the local **DataWriter** (p. 499).

6.36.2.35 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** (p. 499).

6.36.2.36 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** (p. 499).

6.36.2.37 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** (p. 499).

6.36.2.38 InstanceHandle_t DDS::DataWriterProtocolStatus::first_unacknowledged_sample_subscription_handle

The handle of a remote **DataReader** (p. 433) that has not acknowledged the first unacknowledged sample of the local **DataWriter** (p. 499).

6.36.2.39 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. 532) is set.

Sequence (p. 1163) number of the first sample kept in the DataWriters queue whose keep_duration (applied when **DDS::DataWriterProtocolQosPolicy::disable_positive_acks** (p. 532) is set) has not yet elapsed.

6.37 DDS::DataWriterQos Class Reference

QoS policies supported by a **DDS::DataWriter** (p. 499) entity.

`#include <managed_publication.h>`

Public Attributes

- ^ **DurabilityQosPolicy durability**
  Durability policy, ***DURABILITY*** *(p. 276).*
- ^ **DurabilityServiceQosPolicy durability_service**
  DurabilityService policy, ***DURABILITY_SERVICE*** *(p. 297).*
- ^ **DeadlineQosPolicy deadline**
  Deadline policy, ***DEADLINE*** *(p. 281).*
- ^ **LatencyBudgetQosPolicy latency_budget**
  Latency budget policy, ***LATENCY_BUDGET*** *(p. 282).*
- ^ **LivelinessQosPolicy liveliness**
  Liveliness policy, ***LIVELINESS*** *(p. 286).*
- ^ **ReliabilityQosPolicy reliability**
  Reliability policy, ***RELIABILITY*** *(p. 290).*
- ^ **DestinationOrderQosPolicy destination_order**
  Destination order policy, ***DESTINATION_ORDER*** *(p. 292).*
- ^ **HistoryQosPolicy history**
  History policy, ***HISTORY*** *(p. 294).*
- ^ **ResourceLimitsQosPolicy resource_limits**
  Resource limits policy, ***RESOURCE_LIMITS*** *(p. 298).*
- ^ **TransportPriorityQosPolicy transport_priority**
  Transport priority policy, ***TRANSPORT_PRIORITY*** *(p. 300).*
- ^ **LifespanQosPolicy lifespan**
  Lifespan policy, ***LIFESPAN*** *(p. 301).*
- ^ **UserDataQosPolicy^ user_data**
  User data policy, ***USER_DATA*** *(p. 273).*

^ **OwnershipQosPolicy ownership**

Ownership policy, ***OWNERSHIP*** *(*p. *283).*

^ **OwnershipStrengthQosPolicy ownership_strength**

Ownership strength policy, ***OWNERSHIP_STRENGTH*** *(*p. *285).*

^ **WriterDataLifecycleQosPolicy writer_data_lifecycle**

Writer data lifecycle policy, ***WRITER_DATA_LIFECYCLE*** *(*p. *302).*

^ **DataWriterResourceLimitsQosPolicy writer_resource_limits**

<<**eXtension**>> (p. 174) ***DDS::DataWriter*** *(*p. *499) protocol policy,* ***DATA_WRITER_PROTOCOL*** *(*p. *338)*

^ **DataWriterProtocolQosPolicy protocol**

<<**eXtension**>> (p. 174) ***DDS::DataWriter*** *(*p. *499) protocol policy,* ***DATA_WRITER_PROTOCOL*** *(*p. *338)*

^ **TransportSelectionQosPolicy**^ **transport_selection**

<<**eXtension**>> (p. 174) *Transport plugin selection policy,* ***TRANSPORT_SELECTION*** *(*p. *308).*

^ **TransportUnicastQosPolicy**^ **unicast**

<<**eXtension**>> (p. 174) *Unicast transport policy,* ***TRANSPORT_UNICAST*** *(*p. *309).*

^ **PublishModeQosPolicy**^ **publish_mode**

<<**eXtension**>> (p. 174) *Publish mode policy,* ***PUBLISH_MODE*** *(*p. *344).*

^ **PropertyQosPolicy**^ **property_qos**

<<**eXtension**>> (p. 174) *Property policy,* ***PROPERTY*** *(*p. *357).*

^ **BatchQosPolicy batch**

<<**eXtension**>> (p. 174) *Batch policy,* ***BATCH*** *(*p. *353).*

^ **MultiChannelQosPolicy**^ **multi_channel**

<<**eXtension**>> (p. 174) *Multi channel policy,* ***MULTICHANNEL*** *(*p. *356).*

^ **TypeSupportQosPolicy type_support**

<<**eXtension**>> (p. 174) *Type support data,* ***TYPESUPPORT*** *(*p. *350).*

6.37.1 Detailed Description

QoS policies supported by a **DDS::DataWriter** (p. 499) entity.

You must set certain members in a consistent manner:

- DDS::DataWriterQos::history.depth <= DDS::DataWriterQos::resource_limits.max_samples_per_instance

- DDS::DataWriterQos::resource_limits.max_samples_per_instance <= DDS::DataWriterQos::resource_limits.max_samples

- DDS::DataWriterQos::resource_limits.initial_samples <= DDS::DataWriterQos::resource_limits.max_samples

- DDS::DataWriterQos::resource_limits.initial_instances <= DDS::DataWriterQos::resource_limits.max_instances

- length of DDS::DataWriterQos::user_data.value <= **DDS::DomainParticipantQos::resource_limits** (p. 686) .writer_user_data_max_length

If any of the above are not true, **DDS::DataWriter::set_qos** (p. 513) and **DDS::DataWriter::set_qos_with_profile** (p. 514) and **DDS::Publisher::set_default_datawriter_qos** (p. 1049) and **DDS::Publisher::set_default_datawriter_qos_with_profile** (p. 1050) will fail with **DDS::Retcode_InconsistentPolicy** (p. 1119) and **DDS::Publisher::create_datawriter** (p. 1053) and **DDS::Publisher::create_datawriter_with_profile** (p. 1055) and will return NULL.

Entity:

DDS::DataWriter (p. 499)

See also:

QoS Policies (p. 260) allowed ranges within each Qos.

6.37.2 Member Data Documentation

6.37.2.1 DurabilityQosPolicy DDS::DataWriterQos::durability

Durability policy, **DURABILITY** (p. 276).

6.37.2.2 DurabilityServiceQosPolicy DDS::DataWriterQos::durability_service

DurabilityService policy, **DURABILITY_SERVICE** (p. 297).

6.37.2.3 DeadlineQosPolicy DDS::DataWriterQos::deadline

Deadline policy, **DEADLINE** (p. 281).

6.37.2.4 LatencyBudgetQosPolicy DDS::DataWriterQos::latency_budget

Latency budget policy, **LATENCY_BUDGET** (p. 282).

6.37.2.5 LivelinessQosPolicy DDS::DataWriterQos::liveliness

Liveliness policy, **LIVELINESS** (p. 286).

6.37.2.6 ReliabilityQosPolicy DDS::DataWriterQos::reliability

Reliability policy, **RELIABILITY** (p. 290).

6.37.2.7 DestinationOrderQosPolicy DDS::DataWriterQos::destination_order

Destination order policy, **DESTINATION_ORDER** (p. 292).

6.37.2.8 HistoryQosPolicy DDS::DataWriterQos::history

History policy, **HISTORY** (p. 294).

6.37.2.9 ResourceLimitsQosPolicy DDS::DataWriterQos::resource_limits

Resource limits policy, **RESOURCE_LIMITS** (p. 298).

6.37.2.10 TransportPriorityQosPolicy DDS::DataWriterQos::transport_priority

Transport priority policy, **TRANSPORT_PRIORITY** (p. 300).

6.37.2.11 LifespanQosPolicy DDS::DataWriterQos::lifespan

Lifespan policy, **LIFESPAN** (p. 301).

6.37.2.12 UserDataQosPolicy ^ DDS::DataWriterQos::user_data

User data policy, **USER_DATA** (p. 273).

6.37.2.13 OwnershipQosPolicy DDS::DataWriterQos::ownership

Ownership policy, **OWNERSHIP** (p. 283).

6.37.2.14 OwnershipStrengthQosPolicy DDS::DataWriterQos::ownership_strength

Ownership strength policy, **OWNERSHIP_STRENGTH** (p. 285).

6.37.2.15 WriterDataLifecycleQosPolicy DDS::DataWriterQos::writer_data_lifecycle

Writer data lifecycle policy, **WRITER_DATA_LIFECYCLE** (p. 302).

6.37.2.16 DataWriterResourceLimitsQosPolicy DDS::DataWriterQos::writer_resource_limits

<<*eXtension*>> *(p. 174)* **DDS::DataWriter** (p. 499) protocol policy, **DATA_WRITER_PROTOCOL** (p. 338)

6.37.2.17 DataWriterProtocolQosPolicy DDS::DataWriterQos::protocol

<<*eXtension*>> *(p. 174)* **DDS::DataWriter** (p. 499) protocol policy, **DATA_WRITER_PROTOCOL** (p. 338)

6.37.2.18 TransportSelectionQosPolicy ^ DDS::DataWriterQos::transport_selection

<<*eXtension*>> *(p. 174)* Transport plugin selection policy, **TRANSPORT_SELECTION** (p. 308).

Specifies the transports available for use by the **DDS::DataWriter** (p. 499).

6.37.2.19 TransportUnicastQosPolicy ^ DDS::DataWriterQos::unicast

<<*eXtension*>> *(p. 174)* Unicast transport policy, **TRANSPORT_-**

UNICAST (p. 309).

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. 433) entities in the domain.

6.37.2.20 PublishModeQosPolicy ^ DDS::DataWriterQos::publish_mode

<<***eXtension***>> *(p. 174)* Publish mode policy, **PUBLISH_MODE** (p. 344).

Determines whether the **DDS::DataWriter** (p. 499) publishes data synchronously or asynchronously and how.

6.37.2.21 PropertyQosPolicy ^ DDS::DataWriterQos::property_qos

<<***eXtension***>> *(p. 174)* Property policy, **PROPERTY** (p. 357).

6.37.2.22 BatchQosPolicy DDS::DataWriterQos::batch

<<***eXtension***>> *(p. 174)* Batch policy, **BATCH** (p. 353).

6.37.2.23 MultiChannelQosPolicy ^ DDS::DataWriterQos::multi_channel

<<***eXtension***>> *(p. 174)* Multi channel policy, **MULTICHANNEL** (p. 356).

6.37.2.24 TypeSupportQosPolicy DDS::DataWriterQos::type_support

<<***eXtension***>> *(p. 174)* Type support data, **TYPESUPPORT** (p. 350).

Optional value that is passed to a type plugin's on_endpoint_attached and serialization functions.

6.38 DDS::DataWriterResourceLimitsQosPolicy Struct Reference

Various settings that configure how a **DDS::DataWriter** (p. 499) allocates and uses physical memory for internal resources.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_datawriterresourcelimits_qos_policy_name** ()

  *Stringified human-readable name for **DDS::DataWriterResourceLimitsQosPolicy** (p. 552).*

Public Attributes

- System::Int32 **initial_concurrent_blocking_threads**

  *The initial number of threads that are allowed to concurrently block on write call on the same **DDS::DataWriter** (p. 499).*

- System::Int32 **max_concurrent_blocking_threads**

  *The maximum number of threads that are allowed to concurrently block on write call on the same **DDS::DataWriter** (p. 499).*

- System::Int32 **max_remote_reader_filters**

  The maximum number of remote readers for which the writer will perform content-based filtering.

- System::Int32 **initial_batches**

  *Represents the initial number of batches a **DDS::DataWriter** (p. 499) will manage.*

- System::Int32 **max_batches**

  *Represents the maximum number of batches a **DDS::DataWriter** (p. 499) will manage.*

- **DataWriterResourceLimitsInstanceReplacementKind instance_replacement**

  Sets the kinds of instances allowed to be replaced when instance resource limits are reached.

Properties

^ System::Boolean **replace_empty_instances** `[get, set]`

Whether or not to replace empty instances during instance replacement.

^ System::Boolean **autoregister_instances** `[get, set]`

Whether or not to automatically register new instances.

6.38.1 Detailed Description

Various settings that configure how a **DDS::DataWriter** (p. 499) allocates and uses physical memory for internal resources.

DataWriters must allocate internal structures to handle the simultaneously blocking of threads trying to call **DDS::TypedDataWriter::write** (p. 1376) on the same **DDS::DataWriter** (p. 499), 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. 499). By setting the initial size to the maximum size, you will prevent RTI Data Distribution Service from dynamically allocating any memory after the creation of the **DDS::DataWriter** (p. 499).

This QoS policy is an extension to the DDS standard.

Entity:

DDS::DataWriter (p. 499)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)

6.38.2 Member Data Documentation

6.38.2.1 System::Int32 DDS::DataWriterResourceLimitsQosPolicy::initial_concurrent_blocking_threads

The initial number of threads that are allowed to concurrently block on write call on the same **DDS::DataWriter** (p. 499).

This value only applies if **DDS::HistoryQosPolicy** (p. 898) has its kind set to DDS::HistoryQosPolicyKind::KEEP_ALL_HISTORY_QOS and **DDS::ReliabilityQosPolicy::max_blocking_time** (p. 1097) is > 0.

[default] 1

[range] [1, 10000], <= max_concurrent_blocking_threads

6.38.2.2 System::Int32 DDS::DataWriterResourceLimitsQosPolicy::max_concurrent_blocking_threads

The maximum number of threads that are allowed to concurrently block on write call on the same **DDS::DataWriter** (p. 499).

This value only applies if **DDS::HistoryQosPolicy** (p. 898) has its kind set to DDS::HistoryQosPolicyKind::KEEP_ALL_HISTORY_QOS and **DDS::ReliabilityQosPolicy::max_blocking_time** (p. 1097) is > 0.

[default] DDS::LENGTH_UNLIMITED

[range] [1, 10000] or DDS::LENGTH_UNLIMITED, >= initial_concurrent_blocking_threads

6.38.2.3 System::Int32 DDS::DataWriterResourceLimitsQosPolicy::max_remote_reader_filters

The maximum number of remote readers for which the writer will perform content-based filtering.

[default] 32

[range] [0, 32]

6.38.2.4 System::Int32 DDS::DataWriterResourceLimitsQosPolicy::initial_batches

Represents the initial number of batches a **DDS::DataWriter** (p. 499) will manage.

[default] 8

[range] [1,100 million]

See also:

DDS::BatchQosPolicy (p. 376)

6.38.2.5 System::Int32 DDS::DataWriterResourceLimitsQosPolicy::max_batches

Represents the maximum number of batches a **DDS::DataWriter** (p. 499) will manage.

[**default**] DDS::LENGTH_UNLIMITED

When batching is enabled, the maximum number of samples that a **DDS::DataWriter** (p. 499) can store is limited by this value and **DDS::ResourceLimitsQosPolicy::max_samples** (p. 1112).

[**range**] [1,100 million] or DDS::LENGTH_UNLIMITED >= DDS_RtpsReliableWriterProtocol_t::heartbeats_per_max_samples if batching is enabled

See also:

DDS::BatchQosPolicy (p. 376)

6.38.2.6 DataWriterResourceLimitsInstanceReplacementKind DDS::DataWriterResourceLimitsQosPolicy::instance_replacement

Sets the kinds of instances allowed to be replaced when instance resource limits are reached.

When a **DDS::DataWriter** (p. 499)'s number of active instances is greater than **DDS::ResourceLimitsQosPolicy::max_instances** (p. 1112), 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::DataWriterResourceLimitsInstanceReplacementKind::UNREGISTERED_INSTANCE_REPLACEMENT

See also:

DDS::DataWriterResourceLimitsInstanceReplacementKind

6.38.3 Property Documentation

6.38.3.1 System:: Boolean DDS::DataWriterResourceLimitsQosPolicy::replace_empty_instances [get, set]

Whether or not to replace empty instances during instance replacement.

When a **DDS::DataWriter** (p. 499) has more active instances than allowed by **DDS::ResourceLimitsQosPolicy::max_instances** (p. 1112), 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 true, then a **DDS::DataWriter** (p. 499) will first try reclaiming empty instances, before trying to replace whatever is specified by **DDS::DataWriterResourceLimitsQosPolicy::instance_replacement** (p. 555).

[default] false

See also:

DDS::DataWriterResourceLimitsInstanceReplacementKind

6.38.3.2 System:: Boolean DDS::DataWriterResourceLimitsQosPolicy::autoregister_instances [get, set]

Whether or not to automatically register new instances.

[default] true

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:

DDS::TypedDataWriter::write (p. 1376)

6.39 DDS::DeadlineQosPolicy Struct Reference

Expresses the maximum duration (deadline) within which an instance is expected to be updated.

`#include <managed_infrastructure.h>`

Static Public Member Functions

^ static System::String^ **get_deadline_qos_policy_name** ()

Stringified human-readable name for ***DDS::DeadlineQosPolicy*** *(p. 557).*

Public Attributes

^ **Duration_t period**

Duration of the deadline period.

6.39.1 Detailed Description

Expresses the maximum duration (deadline) within which an instance is expected to be updated.

A **DDS::DataReader** (p. 433) 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. 499) indicates that the application commits to write a new value (using the **DDS::DataWriter** (p. 499)) for each instance managed by the **DDS::DataWriter** (p. 499) 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 run time 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. 1258), **DDS::DataReader** (p. 433), **DDS::DataWriter** (p. 499)

Status:

DDS::StatusKind::OFFERED_DEADLINE_MISSED_STATUS,
DDS::StatusKind::REQUESTED_DEADLINE_MISSED_STATUS,
DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS,
DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS

Properties:

RxO (p. 268) = YES
Changeable (p. 269) = **YES** (p. 269)

6.39.2 Usage

This policy is useful for cases where a **DDS::Topic** (p. 1258) 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 Data Distribution Service 'matches' a **DDS::DataWriter** (p. 499) and a **DDS::DataReader** (p. 433) it checks whether the settings are compatible (i.e., *offered deadline <= requested deadline*); if they are not, the two entities are informed (via the **DDS::Listener** (p. 952) or **DDS::Condition** (p. 408) mechanism) of the incompatibility of the QoS settings and communication will not occur.

Assuming that the reader and writer ends have compatible settings, the fulfillment of this contract is monitored by RTI Data Distribution Service and the application is informed of any violations by means of the proper **DDS::Listener** (p. 952) or **DDS::Condition** (p. 408).

6.39.3 Compatibility

The value offered is considered compatible with the value requested if and only if the inequality *offered period <= requested period* holds.

6.39.4 Consistency

The setting of the **DEADLINE** (p. 281) policy must be set consistently with that of the **TIME_BASED_FILTER** (p. 288).

For these two policies to be consistent the settings must be such that *deadline period >= minimum_separation*.

An attempt to set these policies in an inconsistent manner will result

in **DDS::Retcode_InconsistentPolicy** (p. 1119) in **set_qos (abstract)** (p. 846), or the **DDS::Entity** (p. 845) will not be created.

For a **DDS::DataReader** (p. 433), the **DEADLINE** (p. 281) policy and **DDS::TimeBasedFilterQosPolicy** (p. 1254) may interact such that even though the **DDS::DataWriter** (p. 499) is writing samples fast enough to fulfill its commitment to its own deadline, the **DDS::DataReader** (p. 433) may see violations of its deadline. This happens because RTI Data Distribution Service will drop any samples received within the **DDS::TimeBasedFilterQosPolicy::minimum_separation** (p. 1257). To avoid triggering the **DDS::DataReader** (p. 433)'s deadline, even though the matched **DDS::DataWriter** (p. 499) is meeting its own deadline, set the two QoS parameters so that:

reader deadline >= reader minimum_separation + writer deadline

See **DDS::TimeBasedFilterQosPolicy** (p. 1254) for more information about the interactions between deadlines and time-based filters.

See also:

DDS::TimeBasedFilterQosPolicy (p. 1254)

6.39.5 Member Data Documentation

6.39.5.1 Duration_t DDS::DeadlineQosPolicy::period

Duration of the deadline period.

[default] **DDS::Duration_t::DURATION_INFINITE** (p. 253)

[range] [1 nanosec, 1 year] or **DDS::Duration_t::DURATION_INFINITE** (p. 253), >= **DDS::TimeBasedFilterQosPolicy::minimum_separation** (p. 1257)

6.40 DDS::DestinationOrderQosPolicy Struct Reference

Controls how the middleware will deal with data sent by multiple **DDS::DataWriter** (p. 499) entities for the same instance of data (i.e., same **DDS::Topic** (p. 1258) and key).

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_destinationorder_qos_policy_name** ()

  *Stringified human-readable name for **DDS::DestinationOrderQosPolicy** (p. 560).*

Public Attributes

- **DestinationOrderQosPolicyKind kind**

  Specifies the desired kind of destination order.

- **Duration_t source_timestamp_tolerance**

  <<**eXtension**>> (p. 174) *Allowed tolerance between source timestamps of consecutive samples.*

6.40.1 Detailed Description

Controls how the middleware will deal with data sent by multiple **DDS::DataWriter** (p. 499) entities for the same instance of data (i.e., same **DDS::Topic** (p. 1258) and key).

Entity:

DDS::Topic (p. 1258), **DDS::DataReader** (p. 433), **DDS::DataWriter** (p. 499)

Status:

DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS,
DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS

Properties:

RxO (p. 268) = YES
Changeable (p. 269) = **UNTIL ENABLE** (p. 269)

6.40.2 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. 499) entities (which may be associated with different **DDS::Publisher** (p. 1044) entities) running on different nodes.

The default setting, DDS::DestinationOrderQosPolicyKind::BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS, indicates that (assuming the **OWNERSHIP_STRENGTH** (p. 285) 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. 433) in the order in which it was *received* (which may lead to inconsistent final values).

The setting DDS::DestinationOrderQosPolicyKind::BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS indicates that (assuming the **OWNERSHIP_STRENGTH** (p. 285) allows it, within each instance) the `source_timestamp` of the change shall be used to determine the most recent information. That is, data will be delivered by a **DDS::DataReader** (p. 433) in the order in which it was *sent*. If data arrives on the network with a source timestamp that is later than the source timestamp of the last data delivered, the new data will be dropped. This 'by source timestamp' ordering therefore works best when system clocks are relatively synchronized among writing machines.

When using DDS::DestinationOrderQosPolicyKind::BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS, not all data sent by multiple **DDS::DataWriter** (p. 499) entities may be delivered to a **DDS::DataReader** (p. 433) and not all DataReaders will see the same data sent by DataWriters. However, all DataReaders will see the same "final" data when DataWriters "stop" sending data. This is the only setting that, in the case of concurrently publishing **DDS::DataWriter** (p. 499) entities updating the same instance of a shared-ownership topic, ensures all subscribers will end up with the same final value for the instance.

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.

6.40.3 Compatibility

The value offered is considered compatible with the value requested if and only if the inequality *offered kind* >= *requested kind* evaluates to 'TRUE'. For the purposes of this inequality, the values of **DDS::DestinationOrderQosPolicy::kind** (p. 562) are considered ordered such that DDS::DestinationOrderQosPolicyKind::BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS < DDS::DestinationOrderQosPolicyKind::BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS

6.40.4 Member Data Documentation

6.40.4.1 DestinationOrderQosPolicyKind DDS::DestinationOrderQosPolicy::kind

Specifies the desired kind of destination order.

[default] DDS::DestinationOrderQosPolicyKind::BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS,

6.40.4.2 Duration_t DDS::DestinationOrderQosPolicy::source_timestamp_tolerance

*<<**eXtension**>>* *(p. 174)* Allowed tolerance between source timestamps of consecutive samples.

When a **DDS::DataWriter** (p. 499) sets DDS::DestinationOrderQosPolicyKind to DDS::DestinationOrderQosPolicyKind::BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS, when 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. 433) sets DDS::DestinationOrderQosPolicyKind to DDS::DestinationOrderQosPolicyKind::BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS, the **DDS::DataReader** (p. 433) will accept a sample only if the difference between its source timestamp and the reception timestamp is no greater than this tolerance. Otherwise, the sample is rejected.

[default] 100 milliseconds for **DDS::DataWriter** (p. 499), 30 seconds for **DDS::DataReader** (p. 433)

6.41 DDS::DiscoveryConfigQosPolicy Class Reference

Settings for discovery configuration.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_discoveryconfig_qos_policy_name** ()

  *Stringified human-readable name for **DDS::DiscoveryConfigQosPolicy*** (p. 563).

Public Attributes

- **Duration_t participant_liveliness_lease_duration**

  The liveliness lease duration for the participant.

- **Duration_t participant_liveliness_assert_period**

  The period to assert liveliness for the participant.

- **RemoteParticipantPurgeKind remote_participant_purge_kind**

  The participant's behavior for maintaining knowledge of remote participants (and their contained entities) with which discovery communication has been lost.

- **Duration_t max_liveliness_loss_detection_period**

  The maximum amount of time between when a remote entity stops maintaining its liveliness and when the matched local entity realizes that fact.

- System::Int32 **initial_participant_announcements**

  The number of initial announcements sent when a participant is first enabled or when a remote participant is newly discovered.

- **Duration_t min_initial_participant_announcement_period**

  The minimum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.

- **Duration_t max_initial_participant_announcement_period**

  The maximum period between initial announcements when a participant is first enabled or when a remote participant is newly discovered.

- **BuiltinTopicReaderResourceLimits_t participant_reader_resource_limits**

  Resource limits.

- **RtpsReliableReaderProtocol_t publication_reader**

  RTPS protocol-related configuration settings for a built-in publication reader.

- **BuiltinTopicReaderResourceLimits_t publication_reader_resource_limits**

  Resource limits.

- **RtpsReliableReaderProtocol_t subscription_reader**

  RTPS protocol-related configuration settings for a built-in subscription reader.

- **BuiltinTopicReaderResourceLimits_t subscription_reader_resource_limits**

  Resource limits.

- **RtpsReliableWriterProtocol_t publication_writer**

  RTPS protocol-related configuration settings for a built-in publication writer.

- **RtpsReliableWriterProtocol_t subscription_writer**

  RTPS protocol-related configuration settings for a built-in subscription writer.

- System::Int32 **builtin_discovery_plugins**

  The kind mask for built-in discovery plugins.

- **RtpsReliableReaderProtocol_t participant_message_reader**

  RTPS protocol-related configuration settings for a built-in participant message reader.

- **RtpsReliableWriterProtocol_t participant_message_writer**

  RTPS protocol-related configuration settings for a built-in participant message writer.

6.41.1 Detailed Description

Settings for discovery configuration.

This QoS policy is an extension to the DDS standard.

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. 577)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)

6.41.2 Member Data Documentation

6.41.2.1 Duration_t DDS::DiscoveryConfigQosPolicy::participant_liveliness_lease_duration

The liveliness lease duration for the participant.

This is the same as the expiration time of the **DomainParticipant** (p. 577) 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. 565).

[default] 100 seconds

[range] [1 nanosec,1 year], > participant_liveliness_assert_period

6.41.2.2 Duration_t DDS::DiscoveryConfigQosPolicy::participant_liveliness_assert_period

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. 565).

[default] 30 seconds

[range] [1 nanosec,1 year), < participant_liveliness_lease_duration

6.41.2.3 RemoteParticipantPurgeKind DDS::DiscoveryConfigQosPolicy::remote_participant_purge_kind

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::RemoteParticipantPurgeKind::LIVELINESS_BASED_REMOTE_PARTICIPANT_PURGE. However, DDS::RemoteParticipantPurgeKind::NO_REMOTE_PARTICIPANT_PURGE 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.

[default] DDS::RemoteParticipantPurgeKind::LIVELINESS_BASED_REMOTE_PARTICIPANT_PURGE

6.41.2.4 Duration_t DDS::DiscoveryConfigQosPolicy::max_liveliness_loss_detection_period

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 that 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. 433) will learn of the loss of liveliness of a matched **DDS::DataWriter** (p. 499) 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**] [0, 1 year]

6.41.2.5 System::Int32 DDS::DiscoveryConfigQosPolicy::initial_participant_announcements

The number of initial announcements sent when a participant is first enabled or when a remote participant is newly discovered.

Also, when a new remote participant appears, the local participant can announce itself to the peers multiple times controlled by this parameter.

[**default**] 5

[**range**] [0,1 million]

6.41.2.6 Duration_t DDS::DiscoveryConfigQosPolicy::min_initial_participant_announcement_period

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. 568) is introduced in between initial announcements when a new remote participant is discovered.

The setting of **DDS::DiscoveryConfigQosPolicy::min_initial_participant_announcement_period** (p. 567) must be consistent with **DDS::DiscoveryConfigQosPolicy::max_initial_participant_announcement_period** (p. 568). For these two values to be consistent, they must verify that:

DDS::DiscoveryConfigQosPolicy::min_initial_participant_announcement_period (p. 567) <= **DDS::DiscoveryConfigQosPolicy::max_initial_participant_announcement_period** (p. 568).

[**default**] 1 second

[**range**] [1 nanosec,1 year]

6.41.2.7 Duration_t DDS::DiscoveryConfigQosPolicy::max_initial_participant_announcement_period

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. 567) 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. 568) must be consistent with **DDS::DiscoveryConfigQosPolicy::min_initial_participant_announcement_period** (p. 567). For these two values to be consistent, they must verify that:

DDS::DiscoveryConfigQosPolicy::min_initial_participant_announcement_period (p. 567) <= **DDS::DiscoveryConfigQosPolicy::max_initial_participant_announcement_period** (p. 568).

[default] 1 second

[range] [1 nanosec,1 year]

6.41.2.8 BuiltinTopicReaderResourceLimits_t DDS::DiscoveryConfigQosPolicy::participant_reader_resource_limits

Resource limits.

Resource limit of the built-in topic participant reader. For details, see **DDS::BuiltinTopicReaderResourceLimits_t** (p. 385).

6.41.2.9 RtpsReliableReaderProtocol_t DDS::DiscoveryConfigQosPolicy::publication_reader

RTPS protocol-related configuration settings for a built-in publication reader.

For details, refer to the **DDS::DataReaderQos** (p. 480)

6.41.2.10 BuiltinTopicReaderResourceLimits_t DDS::DiscoveryConfigQosPolicy::publication_reader_resource_limits

Resource limits.

Resource limit of the built-in topic publication reader. For details, see **DDS::BuiltinTopicReaderResourceLimits_t** (p. 385).

6.41.2.11 RtpsReliableReaderProtocol_t DDS::DiscoveryConfigQosPolicy::subscription_reader

RTPS protocol-related configuration settings for a built-in subscription reader.

For details, refer to the **DDS::DataReaderQos** (p. 480)

6.41.2.12 BuiltinTopicReaderResourceLimits_t DDS::DiscoveryConfigQosPolicy::subscription_reader_resource_limits

Resource limits.

Resource limit of the built-in topic subscription reader. For details, see **DDS::BuiltinTopicReaderResourceLimits_t** (p. 385).

6.41.2.13 RtpsReliableWriterProtocol_t DDS::DiscoveryConfigQosPolicy::publication_writer

RTPS protocol-related configuration settings for a built-in publication writer.

For details, refer to the **DDS::DataWriterQos** (p. 546)

6.41.2.14 RtpsReliableWriterProtocol_t DDS::DiscoveryConfigQosPolicy::subscription_writer

RTPS protocol-related configuration settings for a built-in subscription writer.

For details, refer to the **DDS::DataWriterQos** (p. 546)

6.41.2.15 System::Int32 DDS::DiscoveryConfigQosPolicy::builtin_discovery_plugins

The kind mask for built-in discovery plugins.

There are several built-in discovery plugin. This mask enables the different plugins. Any plugin not enabled will not be created.

[**default**] DDS::DiscoveryConfigBuiltinPluginKind::DISCOVERYCONFIG_BUILTIN_SDP

6.41.2.16 RtpsReliableReaderProtocol_t DDS::DiscoveryConfigQosPolicy::participant_message_reader

RTPS protocol-related configuration settings for a built-in participant message reader.

For details, refer to the **DDS::DataReaderQos** (p. 480)

6.41.2.17 RtpsReliableWriterProtocol_t DDS::DiscoveryConfigQosPolicy::participant_message_writer

RTPS protocol-related configuration settings for a built-in participant message writer.

For details, refer to the **DDS::DataWriterQos** (p. 546)

6.42 DDS::DiscoveryQosPolicy Class Reference

Configures the mechanism used by the middleware to automatically discover and connect with new remote applications.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_discovery_qos_policy_name** ()

  Stringified human-readable name for ***DDS::DiscoveryQosPolicy*** *(*p. 571*).*

Public Attributes

- **StringSeq^ enabled_transports**

  The transports available for use by the Discovery mechanism.

- **StringSeq^ initial_peers**

  Specifies the multicast group addresses on which discovery-related **metatraffic** *can be received by the domain participant.*

- **StringSeq^ multicast_receive_addresses**

  Specifies the multicast group addresses on which discovery-related **metatraffic** *can be received by the domain participant.*

- System::Int32 **metatraffic_transport_priority**

  The transport priority to use for the Discovery meta-traffic.

Properties

- System::Boolean **accept_unknown_peers** `[get, set]`

  Whether to accept a new participant that is not in the initial peers list.

6.42.1 Detailed Description

Configures the mechanism used by the middleware to automatically discover and connect with new remote applications.

Entity:

DDS::DomainParticipant (p. 577)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)

6.42.2 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. 312)
DDS::DiscoveryConfigQosPolicy (p. 563)

6.42.3 Member Data Documentation

6.42.3.1 StringSeq ^ DDS::DiscoveryQosPolicy::enabled_transports

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. 433) and **DDS::DataWriter** (p. 499)).

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.

Alias names for the builtin transports are defined in **TRANSPORT_BUILTIN** (p. 321).

[default] Empty sequence. All the transports available to the **DomainParticipant** (p. 577) are available for use by the Discovery mechanism.

[range] **Sequence** (p. 1163) of non-null,non-empty strings.

6.42.3.2 StringSeq ^ DDS::DiscoveryQosPolicy::initial_peers

Specifies the multicast group addresses on which discovery-related **metatraffic** can be received by the domain participant.

The multicast group addresses on which the Discovery mechanism will listen for meta-traffic.

Each element of this list must be a valid multicast address (IPv4 or IPv6) in the proper format (see **Address Format** (p. 314)).

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. 573) 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. 573), starting at element 0. They will be stored in the order they appear NDDS_DISCOVERY_PEERS.

Note: Currently, RTI Data Distribution Service will only listen for discovery traffic on the first multicast address (element 0) in **DDS::DiscoveryQosPolicy::multicast_receive_addresses** (p. 573).

[**default**] See **NDDS_DISCOVERY_PEERS** (p. 312)

[**range**] **Sequence** (p. 1163) 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. 314)

6.42.3.3 StringSeq ^ DDS::DiscoveryQosPolicy::multicast_receive_addresses

Specifies the multicast group addresses on which discovery-related **metatraffic** can be received by the domain participant.

The multicast group addresses on which the Discovery mechanism will listen for meta-traffic.

Each element of this list must be a valid multicast address (IPv4 or IPv6) in the proper format (see **Address Format** (p. 314)).

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. 573) 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. 573), starting at element 0. They will be stored in the order they appear NDDS_DISCOVERY_PEERS.

Note: Currently, RTI Data Distribution Service will only listen for discovery traffic on the first multicast address (element 0) in **DDS::DiscoveryQosPolicy::multicast_receive_addresses** (p. 573).

[default] See **NDDS_DISCOVERY_PEERS** (p. 312)

[range] **Sequence** (p. 1163) 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. 314)

6.42.3.4 System::Int32 DDS::DiscoveryQosPolicy::metatraffic_transport_priority

The transport priority to use for the Discovery meta-traffic.

The discovery metatraffic will be sent by the built-in **DDS::DataWriter** (p. 499) using this transport priority.

[default] 0

6.42.4 Property Documentation

6.42.4.1 System:: Boolean DDS::DiscoveryQosPolicy::accept_unknown_peers `[get, set]`

Whether to accept a new participant that is not in the initial peers list.

If false, the participant will only communicate with those in the initial peers list and those added via **DDS::DomainParticipant::add_peer()** (p. 644).

If true, the participant will also communicate with all discovered remote participants.

Note: If accept_unknown_peers is false 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. 312).

[**default**] true

6.43 DDS::DomainEntity Class Reference

<<*interface*>> *(p. 175)* Abstract base class for all DDS entities except for the **DDS::DomainParticipant** (p. 577).

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::DomainEntity::

6.43.1 Detailed Description

<<*interface*>> *(p. 175)* Abstract base class for all DDS entities except for the **DDS::DomainParticipant** (p. 577).

Its sole purpose is to *conceptually* express that **DDS::DomainParticipant** (p. 577) is a special kind of **DDS::Entity** (p. 845) that acts as a container of all other **DDS::Entity** (p. 845) but itself cannot contain other **DDS::DomainParticipant** (p. 577).

6.44 DDS::DomainParticipant Class Reference

<<***interface***>> *(p. 175)* Container for all **DDS::DomainEntity** (p. 576) objects.

`#include <managed_domain.h>`

Inheritance diagram for DDS::DomainParticipant::

Public Member Functions

- ^ void **get_default_datawriter_qos** (**DataWriterQos**^ qos)

  <<**eXtension**>> (p. 174) *Copy the default* ***DDS::DataWriterQos*** *(p. 546) values into the provided* ***DDS::DataWriterQos*** *(p. 546) instance.*

- ^ void **set_default_datawriter_qos** (**DataWriterQos**^ qos)

  <<**eXtension**>> (p. 174) *Set the default* ***DataWriterQos*** *(p. 546) values for this* ***DomainParticipant*** *(p. 577).*

- ^ void **set_default_datawriter_qos_with_profile** (System::String^ library_name, System::String^ profile_name)

  <<**eXtension**>> (p. 174) *Set the default* ***DDS::DataWriterQos*** *(p. 546) values for this domain participant based on the input XML QoS profile.*

- ^ void **get_default_datareader_qos** (**DataReaderQos**^ qos)

  <<**eXtension**>> (p. 174) *Copy the default* ***DDS::DataReaderQos*** *(p. 480) values into the provided* ***DDS::DataReaderQos*** *(p. 480) instance.*

- ^ void **set_default_datareader_qos** (**DataReaderQos**^ qos)

  <<**eXtension**>> (p. 174) *Set the default* ***DDS::DataReaderQos*** *(p. 480) values for this domain participant.*

- ^ void **set_default_datareader_qos_with_profile** (System::String^ library_name, System::String^ profile_name)

  <<**eXtension**>> (p. 174) *Set the default* ***DDS::DataReaderQos*** *(p. 480) values for this* ***DomainParticipant*** *(p. 577) based on the input XML QoS profile.*

- ^ void **get_default_flowcontroller_property** (**FlowControllerProperty_t**^ prop)

  <<**eXtension**>> (p. 174) *Copies the default* ***DDS::FlowControllerProperty_t*** *(p. 871) values for this domain participant into the given* ***DDS::FlowControllerProperty_t*** *(p. 871) instance.*

^ void **set_default_flowcontroller_property** (**FlowControllerProperty_t**^ prop)

<<**eXtension**>> (p. 174) *Set the default **DDS::FlowControllerProperty_t** (p. 871) values for this domain participant.*

^ void **register_contentfilter** (System::String^ filter_name, ContentFilter^ contentfilter)

<<**eXtension**>> (p. 174) *Register a content filter which can be used to create a **DDS::ContentFilteredTopic** (p. 419).*

^ ContentFilter^ **lookup_contentfilter** (System::String^ filter_name)

<<**eXtension**>> (p. 174) *Lookup a content filter previously registered with **DDS::DomainParticipant::register_contentfilter** (p. 593).*

^ void **unregister_contentfilter** (System::String^ filter_name)

<<**eXtension**>> (p. 174) *Unregister a content filter previously registered with **DDS::DomainParticipant::register_contentfilter** (p. 593).*

^ System::String^ **get_default_library** ()

<<**eXtension**>> (p. 174) *Gets the default XML library associated with a **DDS::DomainParticipant** (p. 577).*

^ System::String^ **get_default_profile** ()

<<**eXtension**>> (p. 174) *Gets the default XML profile associated with a **DDS::DomainParticipant** (p. 577).*

^ System::String^ **get_default_profile_library** ()

<<**eXtension**>> (p. 174) *Gets the library where the default XML QoS profile is contained for a **DDS::DomainParticipant** (p. 577).*

^ void **set_default_library** (System::String^ library_name)

<<**eXtension**>> (p. 174) *Sets the default XML library for a **DDS::DomainParticipant** (p. 577).*

^ void **set_default_profile** (System::String^ library_name, System::String^ profile_name)

<<**eXtension**>> (p. 174) *Sets the default XML profile for a **DDS::DomainParticipant** (p. 577).*

^ **Publisher**^ **get_implicit_publisher** ()

<<**eXtension**>> (p. 174) *Returns the implicit **DDS::Publisher** (p. 1044). If an implicit **Publisher** (p. 1044) does not already exist, this creates one.*

^ **Subscriber**^ **get_implicit_subscriber** ()

<<**eXtension**>> (p. 174) *Returns the implicit* ***DDS::Subscriber*** (p. *1201*). *If an implicit* ***Subscriber*** (p. *1201*) *does not already exist, this creates one.*

^ **DataWriter**^ **create_datawriter** (**Topic**^ topic, **DataWriterQos**^ qos, **DataWriterListener**^ listener, **StatusMask** mask)

<<**eXtension**>> (p. 174) *Creates a* ***DDS::DataWriter*** (p. *499*) *that will be attached and belong to the implicit* ***DDS::Publisher*** (p. *1044*).

^ **DataWriter**^ **create_datawriter_with_profile** (**Topic**^ topic, System::String^ library_name, System::String^ profile_name, **DataWriterListener**^ listener, **StatusMask** mask)

<<**eXtension**>> (p. 174) *Creates a* ***DDS::DataWriter*** (p. *499*) *using a XML QoS profile that will be attached and belong to the implicit* ***DDS::Publisher*** (p. *1044*).

^ void **delete_datawriter** (**DataWriter**^ %a_datawriter)

<<**eXtension**>> (p. 174) *Deletes a* ***DDS::DataWriter*** (p. *499*) *that belongs to the implicit* ***DDS::Publisher*** (p. *1044*).

^ **DataReader**^ **create_datareader** (**ITopicDescription**^ topic, **DataReaderQos**^ qos, **DataReaderListener**^ listener, **StatusMask** mask)

<<**eXtension**>> (p. 174) *Creates a* ***DDS::DataReader*** (p. *433*) *that will be attached and belong to the implicit* ***DDS::Subscriber*** (p. *1201*).

^ **DataReader**^ **create_datareader_with_profile** (**ITopicDescription**^ topic, System::String^ library_name, System::String^ profile_name, **DataReaderListener**^ listener, **StatusMask** mask)

<<**eXtension**>> (p. 174) *Creates a* ***DDS::DataReader*** (p. *433*) *using a XML QoS profile that will be attached and belong to the implicit* ***DDS::Subscriber*** (p. *1201*).

^ void **delete_datareader** (**DataReader**^ %a_datareader)

<<**eXtension**>> (p. 174) *Deletes a* ***DDS::DataReader*** (p. *433*) *that belongs to the implicit* ***DDS::Subscriber*** (p. *1201*).

^ void **get_default_topic_qos** (**TopicQos**^ qos)

Copies the default ***DDS::TopicQos*** (p. *1280*) *values for this domain participant into the given* ***DDS::TopicQos*** (p. *1280*) *instance.*

^ void **set_default_topic_qos** (**TopicQos**^ qos)

*Set the default **DDS::TopicQos** (p. 1280) values for this domain participant.*

^ void **set_default_topic_qos_with_profile** (System::String^ library_name, System::String^ profile_name)

<<**eXtension**>> (p. 174) *Set the default **DDS::TopicQos** (p. 1280) values for this domain participant based on the input XML QoS profile.*

^ void **get_default_publisher_qos** (**PublisherQos**^ qos)

*Copy the default **DDS::PublisherQos** (p. 1074) values into the provided **DDS::PublisherQos** (p. 1074) instance.*

^ void **set_default_publisher_qos** (**PublisherQos**^ qos)

*Set the default **DDS::PublisherQos** (p. 1074) values for this **DomainParticipant** (p. 577).*

^ void **set_default_publisher_qos_with_profile** (System::String^ library_name, System::String^ profile_name)

<<**eXtension**>> (p. 174) *Set the default **DDS::PublisherQos** (p. 1074) values for this **DomainParticipant** (p. 577) based on the input XML QoS profile.*

^ void **get_default_subscriber_qos** (**SubscriberQos**^ qos)

*Copy the default **DDS::SubscriberQos** (p. 1230) values into the provided **DDS::SubscriberQos** (p. 1230) instance.*

^ void **set_default_subscriber_qos** (**SubscriberQos**^ qos)

*Set the default **DDS::SubscriberQos** (p. 1230) values for this Domain{articipant.*

^ void **set_default_subscriber_qos_with_profile** (System::String^ library_name, System::String^ profile_name)

<<**eXtension**>> (p. 174) *Set the default **DDS::SubscriberQos** (p. 1230) values for this **DomainParticipant** (p. 577) based on the input XML QoS profile.*

^ **Publisher**^ **create_publisher** (**PublisherQos**^ qos, **PublisherListener**^ listener, **StatusMask** mask)

*Creates a **DDS::Publisher** (p. 1044) with the desired QoS policies and attaches to it the specified **DDS::PublisherListener** (p. 1069).*

^ **Publisher**^ **create_publisher_with_profile** (System::String^ library_name, System::String^ profile_name, **PublisherListener**^ listener, **StatusMask** mask)

<<**eXtension**>> (p. 174) *Creates a new* ***DDS::Publisher*** *(p. 1044) object using the* ***DDS::PublisherQos*** *(p. 1074) associated with the input XML QoS profile.*

^ void **delete_publisher** (**Publisher**^ %p)

Deletes an existing ***DDS::Publisher*** *(p. 1044).*

^ **Subscriber**^ **create_subscriber** (**SubscriberQos**^ qos, **SubscriberListener**^ listener, **StatusMask** mask)

Creates a ***DDS::Subscriber*** *(p. 1201) with the desired QoS policies and attaches to it the specified* ***DDS::SubscriberListener*** *(p. 1226).*

^ **Subscriber**^ **create_subscriber_with_profile** (System::String^ library_name, System::String^ profile_name, **SubscriberListener**^ listener, **StatusMask** mask)

<<**eXtension**>> (p. 174) *Creates a new* ***DDS::Subscriber*** *(p. 1201) object using the* ***DDS::PublisherQos*** *(p. 1074) associated with the input XML QoS profile.*

^ void **delete_subscriber** (**Subscriber**^ %s)

Deletes an existing ***DDS::Subscriber*** *(p. 1201).*

^ void **get_publishers** (**PublisherSeq**^ publishers)

<<**eXtension**>> (p. 174) *Allows the application to access all the publishers the participant has.*

^ void **get_subscribers** (**SubscriberSeq**^ subscribers)

<<**eXtension**>> (p. 174) *Allows the application to access all the subscribers the participant has.*

^ **Topic**^ **create_topic** (System::String^ topic_name, System::String^ type_name, **TopicQos**^ qos, **TopicListener**^ listener, **StatusMask** mask)

Creates a ***DDS::Topic*** *(p. 1258) with the desired QoS policies and attaches to it the specified* ***DDS::TopicListener*** *(p. 1278).*

^ **Topic**^ **create_topic_with_profile** (System::String^ topic_name, System::String^ type_name, System::String^ library_name, System::String^ profile_name, **TopicListener**^ listener, **StatusMask** mask)

<<**eXtension**>> (p. 174) *Creates a new* ***DDS::Topic*** *(p. 1258) object using the* ***DDS::PublisherQos*** *(p. 1074) associated with the input XML QoS profile.*

^ void **delete_topic** (**Topic**^ %topic)

*Deletes a **DDS::Topic** (p. 1258).*

^ **ContentFilteredTopic**^ **create_contentfilteredtopic** (System::String^ name, **Topic**^ related_topic, System::String^ filter_expression, **StringSeq**^ expression_parameters)

*Creates a **DDS::ContentFilteredTopic** (p. 419), that can be used to do content-based subscriptions.*

^ **ContentFilteredTopic**^ **create_contentfilteredtopic_with_filter** (System::String^ name, **Topic**^ related_topic, System::String^ filter_expression, **StringSeq**^ expression_parameters, System::String^ filter_name)

<<**eXtension**>> (p. 174) *Creates a **DDS::ContentFilteredTopic** (p. 419) using the specified filter to do content-based subscriptions.*

^ void **delete_contentfilteredtopic** (**ContentFilteredTopic**^ %a_contentfilteredtopic)

*Deletes a **DDS::ContentFilteredTopic** (p. 419).*

^ **MultiTopic**^ **create_multitopic** (System::String^ name, System::String^ type_name, System::String^ subscription_expression, **StringSeq**^ expression_parameters)

*[**Not supported (optional)**] Creates a **MultiTopic** (p. 984) that can be used to subscribe to multiple topics and combine/filter the received data into a resulting type.*

^ void **delete_multitopic** (**MultiTopic**^ a_multitopic)

*[**Not supported (optional)**] Deletes a **DDS::MultiTopic** (p. 984).*

^ **Topic**^ **find_topic** (System::String^ topic_name, **Duration_t**% timeout)

*Finds an existing (or ready to exist) **DDS::Topic** (p. 1258), based on its name.*

^ **ITopicDescription**^ **lookup_topicdescription** (System::String^ topic_name)

Looks up an existing, locally created DDS::TopicDescription, based on its name.

^ **FlowController**^ **create_flowcontroller** (System::String^ name, **FlowControllerProperty_t**^ prop)

<<**eXtension**>> (p. 174) *Creates a **DDS::FlowController** (p. 867) with the desired property.*

^ void **delete_flowcontroller** (**FlowController**^ %fc)

<<**eXtension**>> (p. 174) *Deletes an existing* ***DDS::FlowController*** *(p. 867).*

^ **FlowController**^ **lookup_flowcontroller** (System::String^ name)

<<**eXtension**>> (p. 174) *Looks up an existing locally-created* ***DDS::FlowController*** *(p. 867), based on its name.*

^ **Subscriber**^ **get_builtin_subscriber** ()

Accesses the **built-in** ***DDS::Subscriber*** *(p. 1201).*

^ void **ignore_participant** (**InstanceHandle_t**% handle)

Instructs RTI Data Distribution Service to locally ignore a remote ***DDS::DomainParticipant*** *(p. 577).*

^ void **ignore_topic** (**InstanceHandle_t**% handle)

Instructs RTI Data Distribution Service to locally ignore a ***DDS::Topic*** *(p. 1258).*

^ void **ignore_publication** (**InstanceHandle_t**% handle)

Instructs RTI Data Distribution Service to locally ignore a publication.

^ void **ignore_subscription** (**InstanceHandle_t**% handle)

Instructs RTI Data Distribution Service to locally ignore a subscription.

^ System::Int32 **get_domain_id** ()

Get the unique domain identifier.

^ void **get_current_time** (**Time_t**% current_time)

Returns the current value of the time.

^ void **assert_liveliness** ()

Manually asserts the liveliness of this ***DDS::DomainParticipant*** *(p. 577).*

^ void **delete_contained_entities** ()

Delete all the entities that were created by means of the "create" operations on the ***DDS::DomainParticipant*** *(p. 577).*

^ void **get_discovered_participants** (**InstanceHandleSeq**^ participant_handles)

Returns list of discovered ***DDS::DomainParticipant*** *(p. 577) s.*

^ void **get_discovered_participant_data** (**ParticipantBuiltinTopicData**^ participant_data, **InstanceHandle_t**% participant_handle)

*Returns **DDS::ParticipantBuiltinTopicData** (p. 1002) for the specified **DDS::DomainParticipant** (p. 577) .*

^ void **get_discovered_topics** (**InstanceHandleSeq**^ topic_handles)

*Returns list of discovered **DDS::Topic** (p. 1258) objects.*

^ void **get_discovered_topic_data** (**TopicBuiltinTopicData**^ topic_data, **InstanceHandle_t**% topic_handle)

*Returns **DDS::TopicBuiltinTopicData** (p. 1268) for the specified **DDS::Topic** (p. 1258).*

^ System::Boolean **contains_entity** (**InstanceHandle_t**% a_handle)

*Completes successfully with true if the referenced **DDS::Entity** (p. 845) is contained by the **DDS::DomainParticipant** (p. 577).*

^ void **set_qos** (**DomainParticipantQos**^ qos)

Change the QoS of this domain participant.

^ void **set_qos_with_profile** (System::String^ library_name, System::String^ profile_name)

<<**eXtension**>> (p. 174) *Change the QoS of this domain participant using the input XML QoS profile.*

^ void **get_qos** (**DomainParticipantQos**^ qos)

Get the participant QoS.

^ void **add_peer** (System::String^ peer_desc_string)

<<**eXtension**>> (p. 174) *Attempt to contact one or more additional peer participants.*

^ void **set_listener** (**DomainParticipantListener**^ l, **StatusMask** mask)

Sets the participant listener.

^ **DomainParticipantListener**^ **get_listener** ()

Get the participant listener.

^ virtual void **enable** () override

*Enables the **DDS::Entity** (p. 845).*

^ virtual **StatusCondition**^ **get_statuscondition** () override

*Allows access to the **DDS::StatusCondition** (p. 1183) associated with the **DDS::Entity** (p. 845).*

^ virtual **StatusMask get_status_changes** () override

*Retrieves the list of communication statuses in the **DDS::Entity*** (p. 845) *that are triggered.*

^ virtual **InstanceHandle_t get_instance_handle** () override

*Allows access to the **DDS::InstanceHandle_t*** (p. 905) *associated with the **DDS::Entity*** (p. 845).

Properties

^ static **PublisherQos**^ **PUBLISHER_QOS_DEFAULT** [get]

*Special value for creating a **DDS::Publisher*** (p. 1044) *with default QoS.*

^ static **SubscriberQos**^ **SUBSCRIBER_QOS_DEFAULT** [get]

*Special value for creating a **DDS::Subscriber*** (p. 1201) *with default QoS.*

^ static **TopicQos**^ **TOPIC_QOS_DEFAULT** [get]

*Special value for creating a **DDS::Topic*** (p. 1258) *with default QoS.*

^ static **FlowControllerProperty_t**^ **FLOW_CONTROLLER_PROPERTY_DEFAULT** [get]

<<**eXtension**>> (p. 174) *Special value for creating a **DDS::FlowController*** (p. 867) *with default property.*

^ static System::String^ **SQLFILTER_NAME** [get]

<<**eXtension**>> (p. 174) *The name of the built-in SQL filter that can be used with ContentFilteredTopics and MultiChannel DataWriters.*

^ static System::String^ **STRINGMATCHFILTER_NAME** [get]

<<**eXtension**>> (p. 174) *The name of the built-in StringMatch filter that can be used with ContentFilteredTopics and MultiChannel DataWriters.*

6.44.1 Detailed Description

<<***interface***>> *(p. 175)* Container for all **DDS::DomainEntity** (p. 576) objects.

The **DomainParticipant** (p. 577) object plays several roles:

- It acts as a container for all other **DDS::Entity** (p. 845) objects.

- It acts as *factory* for the **DDS::Publisher** (p. 1044), **DDS::Subscriber** (p. 1201), **DDS::Topic** (p. 1258) and **DDS::MultiTopic** (p. 984) **DDS::Entity** (p. 845) 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 (**ignore_participant()** (p. 633)), publication (**ignore_publication()** (p. 635)), subscription (**ignore_subscription()** (p. 636)) or topic (**ignore_topic()** (p. 634)).

The following operations may be called even if the **DDS::DomainParticipant** (p. 577) is not enabled. (Operations NOT in this list will fail with the value **DDS::Retcode_NotEnabled** (p. 1121) if called on a disabled **DomainParticipant** (p. 577)).

- Operations defined at the base-class level: **set_qos()** (p. 642), **set_qos_with_profile()** (p. 643), **get_qos()** (p. 643), **set_listener()** (p. 645), **get_listener()** (p. 646), **enable()** (p. 646);

- Factory operations: **create_flowcontroller()** (p. 630), **create_topic()** (p. 621), **create_topic_with_profile()** (p. 623), **create_publisher()** (p. 615), **create_publisher_with_profile()** (p. 616), **create_subscriber()** (p. 618), **create_subscriber_with_profile()** (p. 619), **delete_flowcontroller()** (p. 631), **delete_topic()** (p. 624), **delete_publisher()** (p. 617), **delete_subscriber()** (p. 620), **set_default_flowcontroller_property()** (p. 592), **get_default_flowcontroller_property()** (p. 592), **set_default_topic_qos()** (p. 608), **set_default_topic_qos_with_profile()** (p. 608), **get_default_topic_qos()** (p. 607), **set_default_publisher_qos()** (p. 610), **set_default_publisher_qos_with_profile()** (p. 611), **get_default_publisher_qos()** (p. 610), **set_default_subscriber_qos()** (p. 613), **set_default_subscriber_qos_with_profile()** (p. 614), **get_default_subscriber_qos()** (p. 612), **delete_contained_entities()** (p. 638), **set_default_datareader_qos()** (p. 590), **set_default_datareader_qos_with_profile()** (p. 591), **get_default_datareader_qos()** (p. 589), **set_default_datawriter_qos()** (p. 588), **set_default_datawriter_qos_with_profile()** (p. 589), **get_default_datawriter_qos()** (p. 587), **set_default_library()** (p. 596), **set_default_profile()** (p. 597);

- Operations for looking up topics: **lookup_topicdescription()** (p. 629);
- Operations that access status: **get_statuscondition()** (p. 647), **get_status_changes()** (p. 648).

QoS:

DDS::DomainParticipantQos (p. 683)

Status:

Status Kinds (p. 238)

Listener:

DDS::DomainParticipantListener (p. 675)

See also:

Operations Allowed in Listener Callbacks (p. 954)

Examples:

HelloWorld_publisher.cpp, **HelloWorld_subscriber.cpp**, and **HelloWorldSupport.cpp**.

6.44.2 Member Function Documentation

6.44.2.1 void DDS::DomainParticipant::get_default_datawriter_qos (DataWriterQos^ *qos*)

<<***eXtension***>> *(p. 174)* Copy the default **DDS::DataWriterQos** (p. 546) values into the provided **DDS::DataWriterQos** (p. 546) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **DDS::DomainParticipant::set_default_datawriter_qos** (p. 588), or **DDS::DomainParticipant::set_default_datawriter_qos_with_profile** (p. 589), or else, if the call was never made, the default values listed in **DDS::DataWriterQos** (p. 546).

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

MT Safety:

UNSAFE. It is not safe to retrieve the default **DataWriter** (p. 499) QoS from a DomainPartipant while another thread may be simultaneously calling **DDS::DomainParticipant::set_default_datawriter_qos** (p. 588).

Parameters:

qos *<<**inout**>>* *(p. 176)* Qos to be filled up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.44.2.2 void DDS::DomainParticipant::set_default_datawriter_qos (DataWriterQos^ *qos*)

*<<**eXtension**>>* *(p. 174)* Set the default **DataWriterQos** (p. 546) values for this **DomainParticipant** (p. 577).

This set of default values will be inherited for a newly created **DDS::Publisher** (p. 1044).

Precondition:

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **DDS::Retcode_InconsistentPolicy** (p. 1119)

MT Safety:

UNSAFE. It is not safe to set the default **DataWriter** (p. 499) QoS for a **DomainParticipant** (p. 577) while another thread may be simultaneously calling **DDS::DomainParticipant::set_default_datawriter_qos** (p. 588) or **DDS::DomainParticipant::get_default_datawriter_qos** (p. 587).

Parameters:

qos *<<**in**>>* *(p. 175)* Default qos to be set. The special value **DDS::Publisher::DATAWRITER_QOS_DEFAULT** (p. 80) may be passed as `qos` to indicate that the default QoS should be reset back to the initial values the factory would used if **DDS::DomainParticipant::set_default_datawriter_qos** (p. 588) had never been called. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_InconsistentPolicy** (p. 1119)

6.44.2.3 void DDS::DomainParticipant::set_default_datawriter_qos_with_profile (System::String^ *library_name*, System::String^ *profile_name*)

<<eXtension>> (p. 174) Set the default **DDS::DataWriterQos** (p. 546) 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. 1044).

Precondition:

The **DDS::DataWriterQos** (p. 546) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **DDS::Retcode_InconsistentPolicy** (p. 1119)

MT Safety:

UNSAFE. It is not safe to set the default **DataWriter** (p. 499) QoS for a **DomainParticipant** (p. 577) while another thread may be simultaneously calling **DDS::DomainParticipant::set_default_datawriter_qos** (p. 588) or **DDS::DomainParticipant::get_default_datawriter_qos** (p. 587)

Parameters:

library_name *<<in>> (p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipant::set_default_library** (p. 596)).

profile_name *<<in>> (p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipant::set_default_profile** (p. 597)).

If the input profile cannot be found, the method fails with **DDS::Retcode_Error** (p. 1116).

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_InconsistentPolicy** (p. 1119)

6.44.2.4 void DDS::DomainParticipant::get_default_datareader_qos (DataReaderQos^ *qos*)

<<eXtension>> (p. 174) Copy the default **DDS::DataReaderQos** (p. 480) values into the provided **DDS::DataReaderQos** (p. 480) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **DDS::DomainParticipant::set_default_datareader_qos** (p. 590), or **DDS::DomainParticipant::set_default_datareader_qos_with_profile** (p. 591), or else, if the call was never made, the default values listed in **DDS::DataReaderQos** (p. 480).

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

MT Safety:

UNSAFE. It is not safe to retrieve the default **DataReader** (p. 433) QoS from a **DomainParticipant** (p. 577) while another thread may be simultaneously calling **DDS::DomainParticipant::set_default_datareader_qos** (p. 590).

Parameters:

qos *<<inout>> (p. 176)* Qos to be filled up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.44.2.5 void DDS::DomainParticipant::set_default_datareader_qos (DataReaderQos^ *qos*)

<<eXtension>> (p. 174) Set the default **DDS::DataReaderQos** (p. 480) values for this domain participant.

This set of default values will be inherited for a newly created **DDS::Subscriber** (p. 1201).

Precondition:

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **DDS::Retcode_InconsistentPolicy** (p. 1119)

MT Safety:

UNSAFE. It is not safe to set the default **DataReader** (p. 433) QoS for a **DomainParticipant** (p. 577) while another thread may be simultaneously calling **DDS::DomainParticipant::set_default_datareader_qos** (p. 590) or **DDS::DomainParticipant::get_default_datareader_qos** (p. 589).

Parameters:

qos *<<in>>* *(p. 175)* Default qos to be set. The special value **DDS::Subscriber::DATAREADER_QOS_DEFAULT** (p. 95) may be passed as `qos` to indicate that the default QoS should be reset back to the initial values the factory would used if **DDS::DomainParticipant::set_default_datareader_qos** (p. 590) had never been called. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_InconsistentPolicy** (p. 1119)

6.44.2.6 void DDS::DomainParticipant::set_default_datareader_qos_with_profile (System::String^ *library_name*, System::String^ *profile_name*)

<<eXtension>> *(p. 174)* Set the default **DDS::DataReaderQos** (p. 480) values for this **DomainParticipant** (p. 577) based on the input XML QoS profile.

This set of default values will be inherited for a newly created **DDS::Subscriber** (p. 1201).

Precondition:

The **DDS::DataReaderQos** (p. 480) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **DDS::Retcode_InconsistentPolicy** (p. 1119)

MT Safety:

UNSAFE. It is not safe to set the default **DataReader** (p. 433) QoS for a **DomainParticipant** (p. 577) while another thread may be simultaneously calling **DDS::DomainParticipant::set_default_datareader_qos** (p. 590) or **DDS::DomainParticipant::get_default_datareader_qos** (p. 589).

Parameters:

library_name *<<in>>* *(p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipant::set_default_library** (p. 596)).

profile_name *<<in>>* *(p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipant::set_default_profile** (p. 597)).

If the input profile cannot be found, the method fails with **DDS::Retcode_-Error** (p. 1116).

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_-InconsistentPolicy** (p. 1119)

6.44.2.7 void DDS::DomainParticipant::get_default_-flowcontroller_property (FlowControllerProperty_t^ *prop*)

*<<**eXtension**>>* *(p. 174)* Copies the default **DDS::FlowControllerProperty_t** (p. 871) values for this domain participant into the given **DDS::FlowControllerProperty_t** (p. 871) instance.

The retrieved `property` will match the set of values specified on the last successful call to **DDS::DomainParticipant::set_default_flowcontroller_-property** (p. 592), or else, if the call was never made, the default values listed in **DDS::FlowControllerProperty_t** (p. 871).

MT Safety:

UNSAFE. It is not safe to retrieve the default flow controller properties from a **DomainParticipant** (p. 577) while another thread may be simultaneously calling **DDS::DomainParticipant::set_default_flowcontroller_-property** (p. 592)

Parameters:

prop *<<**in**>>* *(p. 175)* Default property to be retrieved. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::FLOW_CONTROLLER_PROPERTY_DEFAULT
DDS::DomainParticipant::create_flowcontroller (p. 630)

6.44.2.8 void DDS::DomainParticipant::set_default_-flowcontroller_property (FlowControllerProperty_t^ *prop*)

*<<**eXtension**>>* *(p. 174)* Set the default

DDS::FlowControllerProperty_t (p. 871) values for this domain participant.

This default value will be used for newly created **DDS::FlowController** (p. 867) if DDS::FLOW_CONTROLLER_PROPERTY_DEFAULT is specified as the `property` parameter when **DDS::DomainParticipant::create_flowcontroller** (p. 630) is called.

Precondition:

The specified property values must be consistent, or else the operation will have no effect and fail with **DDS::Retcode_InconsistentPolicy** (p. 1119)

MT Safety:

UNSAFE. It is not safe to set the default flow controller properties for a **DomainParticipant** (p. 577) while another thread may be simultaneously calling **DDS::DomainParticipant::set_default_flowcontroller_property** (p. 592) , **DDS::DomainParticipant::get_default_flowcontroller_property** (p. 592) or calling **DDS::DomainParticipant::create_flowcontroller** (p. 630) with DDS::FLOW_CONTROLLER_PROPERTY_DEFAULT as the `qos` parameter.

Parameters:

prop *<<in>>* *(p. 175)* Default property to be set. The special value DDS::FLOW_CONTROLLER_PROPERTY_DEFAULT may be passed as `property` 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. 592) had never been called. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_InconsistentPolicy** (p. 1119)

See also:

DDS::FLOW_CONTROLLER_PROPERTY_DEFAULT
DDS::DomainParticipant::create_flowcontroller (p. 630)

6.44.2.9 void DDS::DomainParticipant::register_contentfilter (System::String^ *filter_name*, ContentFilter^ *contentfilter*) [inline]

<<eXtension>> *(p. 174)* Register a content filter which can be used to create a **DDS::ContentFilteredTopic** (p. 419).

DDS specifies an SQL like content filter for use by content filtered topics. If this filter does not meet the filtering requirements a custom filter can be registered.

Each `filter_name` can only be used to registered a content filter once with a **DDS::DomainParticipant** (p. 577).

Parameters:

filter_name *<<in>> (p. 175)* Name of the filter. The name must be unique within the **DDS::DomainParticipant** (p. 577) and must not exceed 255 characters. Cannot be NULL.

contentfilter *<<in>> (p. 175)* Content filter to be registered. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DomainParticipant::unregister_contentfilter (p. 595)

6.44.2.10 ContentFilter ^ DDS::DomainParticipant::lookup_contentfilter (System::String^ *filter_name*) [inline]

<<eXtension>> (p. 174) Lookup a content filter previously registered with **DDS::DomainParticipant::register_contentfilter** (p. 593).

Parameters:

filter_name *<<in>> (p. 175)* Name of the filter. Cannot be NULL.

Returns:

NULL if the given filter_name has not been previously registered to the **DDS::DomainParticipant** (p. 577) with **DDS::DomainParticipant::register_contentfilter** (p. 593). Otherwise, return the DDS::ContentFilter that has been previosuly registered with the given filter_name.

See also:

DDS::DomainParticipant::register_contentfilter (p. 593)

6.44.2.11 void DDS::DomainParticipant::unregister_contentfilter (System::String^ *filter_name*) [inline]

<<eXtension>> *(p. 174)* Unregister a content filter previously registered with **DDS::DomainParticipant::register_contentfilter** (p. 593).

A `filter_name` can be unregistered only if it has been previously registered to the **DDS::DomainParticipant** (p. 577) with **DDS::DomainParticipant::register_contentfilter** (p. 593).

The unregistration of filter is not allowed if there are any existing **DDS::ContentFilteredTopic** (p. 419) objects that are using the filter. If the operation is called on a filter with existing **DDS::ContentFilteredTopic** (p. 419) objects attached to it, this operation will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123).

If there are still existing discovered **DDS::DataReader** (p. 433) 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. 433) s, finalize method of the filter will be called on those discovered **DDS::DataReader** (p. 433) s before the content filter is unregistered. This means filtering will now be performed on the application that is creating the **DDS::DataReader** (p. 433).

Parameters:

filter_name *<<in>>* *(p. 175)* Name of the filter. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_PreconditionNotMet** (p. 1123)

See also:

DDS::DomainParticipant::register_contentfilter (p. 593)

6.44.2.12 System::String ^ DDS::DomainParticipant::get_default_library ()

<<eXtension>> *(p. 174)* Gets the default XML library associated with a **DDS::DomainParticipant** (p. 577).

Returns:

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

See also:

DDS::DomainParticipant::set_default_library (p. 596)

6.44.2.13 System::String ^ DDS::DomainParticipant::get_default_profile ()

*<<**eXtension**>> (p. 174)* Gets the default XML profile associated with a **DDS::DomainParticipant** (p. 577).

Returns:

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

See also:

DDS::DomainParticipant::set_default_profile (p. 597)

6.44.2.14 System::String ^ DDS::DomainParticipant::get_default_profile_library ()

*<<**eXtension**>> (p. 174)* Gets the library where the default XML QoS profile is contained for a **DDS::DomainParticipant** (p. 577).

The default profile library is automatically set when **DDS::DomainParticipant::set_default_profile** (p. 597) is called.

This library can be different than the **DDS::DomainParticipant** (p. 577) default library (see **DDS::DomainParticipant::get_default_library** (p. 595)).

Returns:

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

See also:

DDS::DomainParticipant::set_default_profile (p. 597)

6.44.2.15 void DDS::DomainParticipant::set_default_library (System::String^ *library_name*)

*<<**eXtension**>> (p. 174)* Sets the default XML library for a **DDS::DomainParticipant** (p. 577).

This method specifies the library that will be used as the default the next time a default library is needed during a call to one of this DomainParticipants 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. 577) inherits the default from the **DDS::DomainParticipantFactory** (p. 649) (see **DDS::DomainParticipantFactory::set_default_library** (p. 657)).

Parameters:

library_name *<<in>>* *(p. 175)* Library name. If library_name is null any previous default is unset.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DomainParticipant::get_default_library (p. 595)

6.44.2.16 void DDS::DomainParticipant::set_default_profile (System::String^ *library_name*, System::String^ *profile_name*)

<<eXtension>> *(p. 174)* Sets the default XML profile for a **DDS::DomainParticipant** (p. 577).

This method specifies the profile that will be used as the default the next time a default **DomainParticipant** (p. 577) profile is needed during a call to one of this DomainParticipants operations. When calling a **DDS::DomainParticipant** (p. 577) method that requires a `profile_name` parameter, you can use NULL to refer to the default profile. (This same information applies to setting a default library.)

If the default profile is not set, the **DDS::DomainParticipant** (p. 577) inherits the default from the **DDS::DomainParticipantFactory** (p. 649) (see **DDS::DomainParticipantFactory::set_default_profile** (p. 658)).

This method does not set the default QoS for entities created by the **DDS::DomainParticipant** (p. 577); for this functionality, use the methods set_default_<entity>_qos_with_profile (you may pass in NULL after having called **set_default_profile()** (p. 597)).

This method does not set the default QoS for newly created DomainParticipants; for this functionality, use **DDS::DomainParticipantFactory::set_default_participant_qos_with_profile** (p. 655).

Parameters:

library_name *<<in>>* *(p. 175)* The library name containing the profile.

profile_name *<<**in**>>* *(p. 175)* The profile name. If profile_name is null any previous default is unset.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DomainParticipant::get_default_profile (p. 596)
DDS::DomainParticipant::get_default_profile_library (p. 596)

6.44.2.17 Publisher ^ DDS::DomainParticipant::get_implicit_publisher ()

*<<**eXtension**>>* *(p. 174)* Returns the implicit **DDS::Publisher** (p. 1044). If an implicit **Publisher** (p. 1044) does not already exist, this creates one.

There can only be one implicit **Publisher** (p. 1044) per **DomainParticipant** (p. 577).

The implicit **Publisher** (p. 1044) is created with **DDS::DomainParticipant::PUBLISHER_QOS_DEFAULT** (p. 38) and no **Listener** (p. 952).

This implicit **Publisher** (p. 1044) will be deleted automatically when the following methods are called: **DDS::DomainParticipant::delete_contained_entities** (p. 638), or **DDS::DomainParticipant::delete_publisher** (p. 617) with the implicit publisher as a parameter. Additionally, when a **DomainParticipant** (p. 577) is deleted, if there are no attached DataWriters that belong to the implicit **Publisher** (p. 1044), the implicit **Publisher** (p. 1044) will be implicitly deleted.

MT Safety:

UNSAFE. It is not safe to create an implicit **Publisher** (p. 1044) while another thread may be simultaneously calling **DDS::DomainParticipant::set_default_publisher_qos** (p. 610).

Returns:

The implicit publisher

See also:

DDS::DomainParticipant::PUBLISHER_QOS_DEFAULT (p. 38)
DDS::DomainParticipant::create_publisher (p. 615)

6.44.2.18 Subscriber ^ DDS::DomainParticipant::get_implicit_subscriber ()

<<*eXtension*>> *(p. 174)* Returns the implicit **DDS::Subscriber** (p. 1201). If an implicit **Subscriber** (p. 1201) does not already exist, this creates one.

There can only be one implicit **Subscriber** (p. 1201) per **DomainParticipant** (p. 577).

The implicit **Subscriber** (p. 1201) is created with **DDS::DomainParticipant::SUBSCRIBER_QOS_DEFAULT** (p. 38) and no **Listener** (p. 952).

This implicit **Subscriber** (p. 1201) will be deleted automatically when the following methods are called: **DDS::DomainParticipant::delete_contained_entities** (p. 638), or **DDS::DomainParticipant::delete_subscriber** (p. 620) with the subscriber as a parameter. Additionally, when a **DomainParticipant** (p. 577) is deleted, if there are no attached DataReaders that belong to the implicit **Subscriber** (p. 1201), the implicit **Subscriber** (p. 1201) 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. 613).

Returns:

The implicit subscriber

See also:

DDS::DomainParticipant::PUBLISHER_QOS_DEFAULT (p. 38)
DDS::DomainParticipant::create_subscriber (p. 618)

6.44.2.19 DataWriter ^ DDS::DomainParticipant::create_datawriter (Topic^ *topic*, DataWriterQos^ *qos*, DataWriterListener^ *listener*, StatusMask *mask*)

<<*eXtension*>> *(p. 174)* Creates a **DDS::DataWriter** (p. 499) that will be attached and belong to the implicit **DDS::Publisher** (p. 1044).

Precondition:

The given **DDS::Topic** (p. 1258) must have been created from the same **DomainParticipant** (p. 577) as the implicit **Publisher** (p. 1044). If it was created from a different **DomainParticipant** (p. 577), this method will fail.

The **DDS::DataWriter** (p. 499) created using this method will be associated with the implicit **Publisher** (p. 1044). This **Publisher** (p. 1044) is automatically created (if it does not exist) using **DDS::DomainParticipant::PUBLISHER_QOS_DEFAULT** (p. 38) when the following methods are called: **DDS::DomainParticipant::create_datawriter** (p. 599), **DDS::DomainParticipant::create_datawriter_with_profile** (p. 601), or **DDS::DomainParticipant::get_implicit_publisher** (p. 598).

MT Safety:

UNSAFE. If **DDS::Publisher::DATAWRITER_QOS_DEFAULT** (p. 80) is used for the `qos` parameter, it is not safe to create the **DataWriter** (p. 499) while another thread may be simultaneously calling **DDS::DomainParticipant::set_default_datawriter_qos** (p. 588).

Parameters:

topic *<<in>>* *(p. 175)* The **DDS::Topic** (p. 1258) that the **DDS::DataWriter** (p. 499) will be associated with. Cannot be NULL.

qos *<<in>>* *(p. 175)* QoS to be used for creating the new **DDS::DataWriter** (p. 499). The special value **DDS::Publisher::DATAWRITER_QOS_DEFAULT** (p. 80) can be used to indicate that the **DDS::DataWriter** (p. 499) should be created with the default **DDS::DataWriterQos** (p. 546) set in the implicit **DDS::Publisher** (p. 1044). The special value DDS::DATAWRITER_QOS_USE_TOPIC_QOS can be used to indicate that the **DDS::DataWriter** (p. 499) should be created with the combination of the default **DDS::DataWriterQos** (p. 546) set on the **DDS::Publisher** (p. 1044) and the **DDS::TopicQos** (p. 1280) of the **DDS::Topic** (p. 1258). Cannot be NULL.

listener *<<in>>* *(p. 175)* The listener of the **DDS::DataWriter** (p. 499).

mask *<<in>>* *(p. 175)*. Changes of communication status to be invoked on the listener.

Returns:

A **DDS::DataWriter** (p. 499) of a derived class specific to the data type associated with the **DDS::Topic** (p. 1258) or NULL if an error occurred.

See also:

DDS::TypedDataWriter (p. 1368)
Specifying QoS on entities (p. 267) for information on setting QoS before entity creation

DDS::DataWriterQos (p. 546) for rules on consistency among QoS
DDS::Publisher::DATAWRITER_QOS_DEFAULT (p. 80)
DDS::DATAWRITER_QOS_USE_TOPIC_QOS
DDS::DomainParticipant::create_datawriter_with_profile (p. 601)
DDS::DomainParticipant::get_default_datawriter_qos (p. 587)
DDS::DomainParticipant::get_implicit_publisher (p. 598)
DDS::Topic::set_qos (p. 1261)
DDS::DataWriter::set_listener (p. 515)

6.44.2.20 DataWriter ^ DDS::DomainParticipant::create_datawriter_with_profile (Topic^ *topic*, System::String^ *library_name*, System::String^ *profile_name*, DataWriterListener^ *listener*, StatusMask *mask*)

<<eXtension>> (p. 174) Creates a **DDS::DataWriter** (p. 499) using a XML QoS profile that will be attached and belong to the implicit **DDS::Publisher** (p. 1044).

Precondition:

The given **DDS::Topic** (p. 1258) must have been created from the same **DomainParticipant** (p. 577) as the implicit **Publisher** (p. 1044). If it was created from a different **DomainParticipant** (p. 577), this method will return NULL.

The **DDS::DataWriter** (p. 499) created using this method will be associated with the implicit **Publisher** (p. 1044). This **Publisher** (p. 1044) is automatically created (if it does not exist) using **DDS::DomainParticipant::PUBLISHER_QOS_DEFAULT** (p. 38) when the following methods are called: **DDS::DomainParticipant::create_datawriter** (p. 599), **DDS::DomainParticipant::create_datawriter_with_profile** (p. 601), or **DDS::DomainParticipant::get_implicit_publisher** (p. 598)

Parameters:

topic *<<in>> (p. 175)* The **DDS::Topic** (p. 1258) that the **DDS::DataWriter** (p. 499) will be associated with. Cannot be NULL.

library_name *<<in>> (p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipant::set_default_library** (p. 596)).

profile_name *<<in>> (p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipant::set_default_profile** (p. 597)).

listener *<<in>>* *(p. 175)* The listener of the **DDS::DataWriter** (p. 499).

mask *<<in>>* *(p. 175)*. Changes of communication status to be invoked on the listener.

Returns:

A **DDS::DataWriter** (p. 499) of a derived class specific to the data type associated with the **DDS::Topic** (p. 1258) or NULL if an error occurred.

See also:

DDS::TypedDataWriter (p. 1368)
Specifying QoS on entities (p. 267) for information on setting QoS before entity creation
DDS::DataWriterQos (p. 546) for rules on consistency among QoS
DDS::DomainParticipant::create_datawriter (p. 599)
DDS::DomainParticipant::get_default_datawriter_qos (p. 587)
DDS::DomainParticipant::get_implicit_publisher (p. 598)
DDS::Topic::set_qos (p. 1261)
DDS::DataWriter::set_listener (p. 515)

6.44.2.21 void DDS::DomainParticipant::delete_datawriter (DataWriter^ % *a_datawriter*)

<<eXtension>> *(p. 174)* Deletes a **DDS::DataWriter** (p. 499) that belongs to the implicit **DDS::Publisher** (p. 1044).

The deletion of the **DDS::DataWriter** (p. 499) will automatically unregister all instances. Depending on the settings of the **WRITER_DATA_LIFECYCLE** (p. 302) QosPolicy, the deletion of the **DDS::DataWriter** (p. 499) may also dispose all instances.

6.44.3 Special Instructions if Using Timestamp APIs and BY_SOURCE_TIMESTAMP Destination Ordering:

If the DataWriters **DDS::DestinationOrderQosPolicy::kind** (p. 562) is DDS::DestinationOrderQosPolicyKind::BY_SOURCE_TIMESTAMP_-DESTINATIONORDER_QOS, calls to **delete_datawriter()** (p. 602) may fail if your application has previously used the with timestamp APIs (write_w_timestamp(), register_instance_w_timestamp(), unregister_instance_-w_timestamp(), or dispose_w_timestamp()) with a timestamp larger (later) than the time at which **delete_datawriter()** (p. 602) is called. To prevent **delete_datawriter()** (p. 602) from failing in this situation, either:

- Change the **WRITER_DATA_LIFECYCLE** (p. 302) QosPolicy so that RTI Data Distribution Service will not autodispose unregistered instances (set **DDS::WriterDataLifecycleQosPolicy::autodispose_unregistered_instances** (p. 1432) to false.) or
- Explicitly call unregister_instance_w_timestamp() for all instances modified with the *_w_timestamp() APIs before calling **delete_datawriter()** (p. 602).

Precondition:

If the **DDS::DataWriter** (p. 499) does not belong to the implicit **DDS::Publisher** (p. 1044), the operation will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123).

Postcondition:

Listener (p. 952) installed on the **DDS::DataWriter** (p. 499) will not be called after this method completes successfully.

Parameters:

a_datawriter *<<in>> (p. 175)* The **DDS::DataWriter** (p. 499) to be deleted.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_PreconditionNotMet** (p. 1123).

See also:

DDS::DomainParticipant::get_implicit_publisher (p. 598)

6.44.3.1 DataReader ^ DDS::DomainParticipant::create_datareader (ITopicDescription^ *topic*, DataReaderQos^ *qos*, DataReaderListener^ *listener*, StatusMask *mask*)

<<eXtension>> (p. 174) Creates a **DDS::DataReader** (p. 433) that will be attached and belong to the implicit **DDS::Subscriber** (p. 1201).

Precondition:

The given DDS::TopicDescription must have been created from the same **DomainParticipant** (p. 577) as the implicit **Subscriber** (p. 1201). If it was created from a different **DomainParticipant** (p. 577), this method will return NULL.

The **DDS::DataReader** (p. 433) created using this method will be associated with the implicit **Subscriber** (p. 1201). This **Subscriber** (p. 1201) is automatically created (if it does not exist) using **DDS::DomainParticipant::SUBSCRIBER_QOS_DEFAULT** (p. 38) when the following methods are called: **DDS::DomainParticipant::create_datareader** (p. 603), **DDS::DomainParticipant::create_datareader_with_profile** (p. 605), or **DDS::DomainParticipant::get_implicit_subscriber** (p. 599).

MT Safety:

UNSAFE. If **DDS::Subscriber::DATAREADER_QOS_DEFAULT** (p. 95) 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. 590).

Parameters:

topic *<<in>>* *(p. 175)* The DDS::TopicDescription that the **DDS::DataReader** (p. 433) will be associated with. Cannot be NULL.

qos *<<in>>* *(p. 175)* The qos of the **DDS::DataReader** (p. 433). The special value **DDS::Subscriber::DATAREADER_QOS_DEFAULT** (p. 95) can be used to indicate that the **DDS::DataReader** (p. 433) should be created with the default **DDS::DataReaderQos** (p. 480) set in the implicit **DDS::Subscriber** (p. 1201). If DDS::TopicDescription is of type **DDS::Topic** (p. 1258) or **DDS::ContentFilteredTopic** (p. 419), the special value DDS::DATAREADER_QOS_USE_TOPIC_QOS can be used to indicate that the **DDS::DataReader** (p. 433) should be created with the combination of the default **DDS::DataReaderQos** (p. 480) set on the implicit **DDS::Subscriber** (p. 1201) and the **DDS::TopicQos** (p. 1280) (in the case of a **DDS::ContentFilteredTopic** (p. 419), the **DDS::TopicQos** (p. 1280) of the related **DDS::Topic** (p. 1258)). if DDS::DATAREADER_QOS_USE_TOPIC_QOS is used, `topic` cannot be a **DDS::MultiTopic** (p. 984). Cannot be NULL.

listener *<<in>>* *(p. 175)* The listener of the **DDS::DataReader** (p. 433).

mask *<<in>>* *(p. 175)*. Changes of communication status to be invoked on the listener.

Returns:

A **DDS::DataReader** (p. 433) of a derived class specific to the data-type associated with the **DDS::Topic** (p. 1258) or NULL if an error occurred.

See also:

DDS::TypedDataReader (p. 1338)
Specifying QoS on entities (p. 267) for information on setting QoS before entity creation
DDS::DataReaderQos (p. 480) for rules on consistency among QoS
DDS::DomainParticipant::create_datareader_with_profile (p. 605)
DDS::DomainParticipant::get_default_datareader_qos (p. 589)
DDS::DomainParticipant::get_implicit_subscriber (p. 599)
DDS::Topic::set_qos (p. 1261)
DDS::DataReader::set_listener (p. 450)

6.44.3.2 DataReader ^ DDS::DomainParticipant::create_datareader_with_profile (ITopicDescription^ *topic*, System::String^ *library_name*, System::String^ *profile_name*, DataReaderListener^ *listener*, StatusMask *mask*)

<<eXtension>> *(p. 174)* Creates a **DDS::DataReader** (p. 433) using a XML QoS profile that will be attached and belong to the implicit **DDS::Subscriber** (p. 1201).

Precondition:

The given DDS::TopicDescription must have been created from the same **DomainParticipant** (p. 577) as the implicit subscriber. If it was created from a different **DomainParticipant** (p. 577), this method will return NULL.

The **DDS::DataReader** (p. 433) created using this method will be associated with the implicit **Subscriber** (p. 1201). This **Subscriber** (p. 1201) is automatically created (if it does not exist) using **DDS::DomainParticipant::SUBSCRIBER_QOS_DEFAULT** (p. 38) when the following methods are called: **DDS::DomainParticipant::create_datareader** (p. 603), **DDS::DomainParticipant::create_datareader_with_profile** (p. 605), or **DDS::DomainParticipant::get_implicit_subscriber** (p. 599)

Parameters:

topic *<<in>>* *(p. 175)* The DDS::TopicDescription that the **DDS::DataReader** (p. 433) will be associated with. Cannot be NULL.

library_name *<<in>>* *(p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use

the default library (see **DDS::DomainParticipant::set_default_library** (p. 596)).

profile_name *<<in>> (p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipant::set_default_profile** (p. 597)).

listener *<<in>> (p. 175)* The listener of the **DDS::DataReader** (p. 433).

mask *<<in>> (p. 175).* Changes of communication status to be invoked on the listener.

Returns:

A **DDS::DataReader** (p. 433) of a derived class specific to the data-type associated with the **DDS::Topic** (p. 1258) or NULL if an error occurred.

See also:

DDS::TypedDataReader (p. 1338)
Specifying QoS on entities (p. 267) for information on setting QoS before entity creation
DDS::DataReaderQos (p. 480) for rules on consistency among QoS
DDS::DomainParticipant::create_datareader (p. 603)
DDS::DomainParticipant::get_default_datareader_qos (p. 589)
DDS::DomainParticipant::get_implicit_subscriber (p. 599)
DDS::Topic::set_qos (p. 1261)
DDS::DataReader::set_listener (p. 450)

6.44.3.3 void DDS::DomainParticipant::delete_datareader (DataReader^ % *a_datareader*)

<<eXtension>> (p. 174) Deletes a **DDS::DataReader** (p. 433) that belongs to the implicit **DDS::Subscriber** (p. 1201).

Precondition:

If the **DDS::DataReader** (p. 433) does not belong to the implicit **DDS::Subscriber** (p. 1201), or if ther are any existing **DDS::ReadCondition** (p. 1084) or **DDS::QueryCondition** (p. 1082) objects that are attached to the **DDS::DataReader** (p. 433), 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_PreconditionNotMet** (p. 1123).

Postcondition:

Listener (p. 952) installed on the **DDS::DataReader** (p. 433) will not be called after this method completes successfully.

Parameters:

a_datareader *<<in>>* *(p. 175)* The **DDS::DataReader** (p. 433) to be deleted.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_PreconditionNotMet** (p. 1123).

See also:

DDS::DomainParticipant::get_implicit_subscriber (p. 599)

6.44.3.4 void DDS::DomainParticipant::get_default_topic_qos (TopicQos^ *qos*)

Copies the default **DDS::TopicQos** (p. 1280) values for this domain participant into the given **DDS::TopicQos** (p. 1280) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **DDS::DomainParticipant::set_default_topic_qos** (p. 608), or else, if the call was never made, the default values listed in **DDS::TopicQos** (p. 1280).

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

MT Safety:

UNSAFE. It is not safe to retrieve the default **Topic** (p. 1258) QoS from a **DomainParticipant** (p. 577) while another thread may be simultaneously calling **DDS::DomainParticipant::set_default_topic_qos** (p. 608)

Parameters:

qos *<<in>>* *(p. 175)* Default qos to be retrieved. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DomainParticipant::TOPIC_QOS_DEFAULT (p. 39)
DDS::DomainParticipant::create_topic (p. 621)

6.44.3.5 void DDS::DomainParticipant::set_default_topic_qos (TopicQos^ *qos*)

Set the default **DDS::TopicQos** (p. 1280) values for this domain participant.

This default value will be used for newly created **DDS::Topic** (p. 1258) if **DDS::DomainParticipant::TOPIC_QOS_DEFAULT** (p. 39) is specified as the `qos` parameter when **DDS::DomainParticipant::create_topic** (p. 621) is called.

Precondition:

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **DDS::Retcode_InconsistentPolicy** (p. 1119)

MT Safety:

UNSAFE. It is not safe to set the default topic QoS for a **DomainParticipant** (p. 577) while another thread may be simultaneously calling **DDS::DomainParticipant::set_default_topic_qos** (p. 608), **DDS::DomainParticipant::get_default_topic_qos** (p. 607) or calling **DDS::DomainParticipant::create_topic** (p. 621) with **DDS::DomainParticipant::TOPIC_QOS_DEFAULT** (p. 39) as the `qos` parameter.

Parameters:

qos *<<in>> (p. 175)* Default qos to be set. The special value **DDS::DomainParticipant::TOPIC_QOS_DEFAULT** (p. 39) may be passed as `qos` to indicate that the default QoS should be reset back to the initial values the factory would used if **DDS::DomainParticipant::set_default_topic_qos** (p. 608) had never been called. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_InconsistentPolicy** (p. 1119)

See also:

DDS::DomainParticipant::TOPIC_QOS_DEFAULT (p. 39)
DDS::DomainParticipant::create_topic (p. 621)

6.44.3.6 void DDS::DomainParticipant::set_default_topic_qos_with_profile (System::String^ *library_name*, System::String^ *profile_name*)

<<eXtension>> (p. 174) Set the default **DDS::TopicQos** (p. 1280) values

for this domain participant based on the input XML QoS profile.

This default value will be used for newly created **DDS::Topic** (p. 1258) if **DDS::DomainParticipant::TOPIC_QOS_DEFAULT** (p. 39) is specified as the `qos` parameter when **DDS::DomainParticipant::create_topic** (p. 621) is called.

Precondition:

The **DDS::TopicQos** (p. 1280) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **DDS::Retcode_InconsistentPolicy** (p. 1119)

MT Safety:

UNSAFE. It is not safe to set the default topic QoS for a **DomainParticipant** (p. 577) while another thread may be simultaneously calling **DDS::DomainParticipant::set_default_topic_qos** (p. 608), **DDS::DomainParticipant::get_default_topic_qos** (p. 607) or calling **DDS::DomainParticipant::create_topic** (p. 621) with **DDS::DomainParticipant::TOPIC_QOS_DEFAULT** (p. 39) as the `qos` parameter.

Parameters:

library_name *<<in>>* *(p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipant::set_default_library** (p. 596)).

profile_name *<<in>>* *(p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipant::set_default_profile** (p. 597)).

If the input profile cannot be found the method fails with **DDS::Retcode_Error** (p. 1116).

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_InconsistentPolicy** (p. 1119)

See also:

DDS::DomainParticipant::TOPIC_QOS_DEFAULT (p. 39)
DDS::DomainParticipant::create_topic_with_profile (p. 623)

6.44.3.7 void DDS::DomainParticipant::get_default_publisher_qos (PublisherQos^ *qos*)

Copy the default **DDS::PublisherQos** (p. 1074) values into the provided **DDS::PublisherQos** (p. 1074) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **DDS::DomainParticipant::set_default_publisher_qos** (p. 610), or **DDS::DomainParticipant::set_default_publisher_qos_with_profile** (p. 611), or else, if the call was never made, the default values listed in **DDS::PublisherQos** (p. 1074).

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

If **DDS::DomainParticipant::PUBLISHER_QOS_DEFAULT** (p. 38) is specified as the `qos` parameter when **DDS::DomainParticipant::create_topic** (p. 621) 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. 610), will be used to create the **DDS::Publisher** (p. 1044).

MT Safety:

UNSAFE. It is not safe to retrieve the default publisher QoS from a **DomainParticipant** (p. 577) while another thread may be simultaneously calling **DDS::DomainParticipant::set_default_publisher_qos** (p. 610)

Parameters:

qos *<<inout>> (p. 176)* Qos to be filled up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DomainParticipant::PUBLISHER_QOS_DEFAULT (p. 38)
DDS::DomainParticipant::create_publisher (p. 615)

6.44.3.8 void DDS::DomainParticipant::set_default_publisher_qos (PublisherQos^ *qos*)

Set the default **DDS::PublisherQos** (p. 1074) values for this **DomainParticipant** (p. 577).

This set of default values will be used for a newly created **DDS::Publisher** (p. 1044) if **DDS::DomainParticipant::PUBLISHER_-**

QOS_DEFAULT (p. 38) is specified as the `qos` parameter when **DDS::DomainParticipant::create_publisher** (p. 615) is called.

Precondition:

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **DDS::Retcode_InconsistentPolicy** (p. 1119)

MT Safety:

UNSAFE. It is not safe to set the default publisher QoS for a **DomainParticipant** (p. 577) while another thread may be simultaneously calling **DDS::DomainParticipant::set_default_publisher_qos** (p. 610), **DDS::DomainParticipant::get_default_publisher_qos** (p. 610) or calling **DDS::DomainParticipant::create_publisher** (p. 615) with **DDS::DomainParticipant::PUBLISHER_QOS_DEFAULT** (p. 38) as the `qos` parameter.

Parameters:

qos *<<in>>* *(p. 175)* Default qos to be set. The special value **DDS::DomainParticipant::PUBLISHER_QOS_DEFAULT** (p. 38) may be passed as `qos` to indicate that the default QoS should be reset back to the initial values the factory would used if **DDS::DomainParticipant::set_default_publisher_qos** (p. 610) had never been called. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_InconsistentPolicy** (p. 1119)

See also:

DDS::DomainParticipant::PUBLISHER_QOS_DEFAULT (p. 38)
DDS::DomainParticipant::create_publisher (p. 615)

6.44.3.9 void DDS::DomainParticipant::set_default_publisher_qos_with_profile (System::String^ *library_name*, System::String^ *profile_name*)

<<eXtension>> *(p. 174)* Set the default **DDS::PublisherQos** (p. 1074) values for this **DomainParticipant** (p. 577) based on the input XML QoS profile.

This set of default values will be used for a newly created **DDS::Publisher** (p. 1044) if **DDS::DomainParticipant::PUBLISHER_QOS_DEFAULT** (p. 38) is specified as the `qos` parameter when **DDS::DomainParticipant::create_publisher** (p. 615) is called.

Precondition:

The **DDS::PublisherQos** (p. 1074) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **DDS::Retcode_InconsistentPolicy** (p. 1119)

MT Safety:

UNSAFE. It is not safe to set the default publisher QoS for a **DomainParticipant** (p. 577) while another thread may be simultaneously calling **DDS::DomainParticipant::set_default_publisher_qos** (p. 610), **DDS::DomainParticipant::get_default_publisher_qos** (p. 610) or calling **DDS::DomainParticipant::create_publisher** (p. 615) with **DDS::DomainParticipant::PUBLISHER_QOS_DEFAULT** (p. 38) as the `qos` parameter.

Parameters:

library_name *<<**in**>>* *(p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipant::set_default_library** (p. 596)).

profile_name *<<**in**>>* *(p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipant::set_default_profile** (p. 597)).

If the input profile cannot be found, the method fails with **DDS::Retcode_Error** (p. 1116).

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_InconsistentPolicy** (p. 1119)

See also:

DDS::DomainParticipant::PUBLISHER_QOS_DEFAULT (p. 38)
DDS::DomainParticipant::create_publisher_with_profile (p. 616)

6.44.3.10 void DDS::DomainParticipant::get_default_subscriber_qos (SubscriberQos^ *qos*)

Copy the default **DDS::SubscriberQos** (p. 1230) values into the provided **DDS::SubscriberQos** (p. 1230) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **DDS::DomainParticipant::set_default_subscriber_qos** (p. 613), or **DDS::DomainParticipant::set_default_subscriber_qos_with_profile** (p. 614), or else, if the call was never made, the default values listed in **DDS::SubscriberQos** (p. 1230).

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

If **DDS::DomainParticipant::SUBSCRIBER_QOS_DEFAULT** (p. 38) is specified as the `qos` parameter when **DDS::DomainParticipant::create_subscriber** (p. 618) 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. 612), will be used to create the **DDS::Subscriber** (p. 1201).

MT Safety:

UNSAFE. It is not safe to retrieve the default **Subscriber** (p. 1201) QoS from a **DomainParticipant** (p. 577) while another thread may be simultaneously calling **DDS::DomainParticipant::set_default_subscriber_qos** (p. 613).

Parameters:

qos *<<inout>>* *(p. 176)* Qos to be filled up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DomainParticipant::SUBSCRIBER_QOS_DEFAULT (p. 38)
DDS::DomainParticipant::create_subscriber (p. 618)

6.44.3.11 void DDS::DomainParticipant::set_default_subscriber_qos (SubscriberQos^ *qos*)

Set the default **DDS::SubscriberQos** (p. 1230) values for this Domain{articipant.

This set of default values will be used for a newly created **DDS::Subscriber** (p. 1201) if **DDS::DomainParticipant::SUBSCRIBER_QOS_DEFAULT** (p. 38) is specified as the `qos` parameter when **DDS::DomainParticipant::create_subscriber** (p. 618) is called.

Precondition:

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **DDS::Retcode_InconsistentPolicy** (p. 1119)

MT Safety:

UNSAFE. It is not safe to set the default **Subscriber** (p. 1201) QoS for a **DomainParticipant** (p. 577) while another thread may be simultaneously calling **DDS::DomainParticipant::set_default_subscriber_qos** (p. 613), **DDS::DomainParticipant::get_default_subscriber_qos** (p. 612) or calling **DDS::DomainParticipant::create_subscriber** (p. 618) with **DDS::DomainParticipant::SUBSCRIBER_QOS_DEFAULT** (p. 38) as the `qos` parameter.

Parameters:

qos *<<in>> (p. 175)* Default qos to be set. The special value **DDS::DomainParticipant::SUBSCRIBER_QOS_DEFAULT** (p. 38) may be passed as `qos` 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. 613) had never been called. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_InconsistentPolicy** (p. 1119)

6.44.3.12 void DDS::DomainParticipant::set_default_subscriber_qos_with_profile (System::String^ *library_name*, System::String^ *profile_name*)

<<eXtension>> (p. 174) Set the default **DDS::SubscriberQos** (p. 1230) values for this **DomainParticipant** (p. 577) based on the input XML QoS profile.

This set of default values will be used for a newly created **DDS::Subscriber** (p. 1201) if **DDS::DomainParticipant::SUBSCRIBER_QOS_DEFAULT** (p. 38) is specified as the `qos` parameter when **DDS::DomainParticipant::create_subscriber** (p. 618) is called.

Precondition:

The **DDS::SubscriberQos** (p. 1230) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **DDS::Retcode_InconsistentPolicy** (p. 1119)

MT Safety:

UNSAFE. It is not safe to set the default **Subscriber** (p. 1201) QoS for a **DomainParticipant** (p. 577) while another thread may be simultaneously calling **DDS::DomainParticipant::set_default_subscriber_qos** (p. 613), **DDS::DomainParticipant::get_default_subscriber_qos** (p. 612) or calling **DDS::DomainParticipant::create_subscriber** (p. 618) with **DDS::DomainParticipant::SUBSCRIBER_QOS_DEFAULT** (p. 38) as the `qos` parameter.

Parameters:

library_name *<<in>>* *(p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipant::set_default_library** (p. 596)).

profile_name *<<in>>* *(p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipant::set_default_profile** (p. 597)).

If the input profile cannot be found, the method fails with **DDS::Retcode_Error** (p. 1116).

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_InconsistentPolicy** (p. 1119)

See also:

DDS::DomainParticipant::SUBSCRIBER_QOS_DEFAULT (p. 38)
DDS::DomainParticipant::create_subscriber_with_profile (p. 619)

6.44.3.13 Publisher ^ DDS::DomainParticipant::create_publisher (PublisherQos^ *qos*, PublisherListener^ *listener*, StatusMask *mask*)

Creates a **DDS::Publisher** (p. 1044) with the desired QoS policies and attaches to it the specified **DDS::PublisherListener** (p. 1069).

Precondition:

The specified QoS policies must be consistent, or the operation will fail and no **DDS::Publisher** (p. 1044) will be created.

MT Safety:

UNSAFE. If **DDS::DomainParticipant::PUBLISHER_QOS_DEFAULT** (p. 38) 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. 610).

Parameters:

qos *<<in>>* *(p. 175)* QoS to be used for creating the new **DDS::Publisher** (p. 1044). The special value **DDS::DomainParticipant::PUBLISHER_QOS_DEFAULT** (p. 38) can be used to indicate that the **DDS::Publisher** (p. 1044) should be created with the default **DDS::PublisherQos** (p. 1074) set in the **DDS::DomainParticipant** (p. 577). Cannot be NULL.

listener *<<in>>* *(p. 175)*. **Listener** (p. 952) to be attached to the newly created **DDS::Publisher** (p. 1044).

mask *<<in>>* *(p. 175)*. Changes of communication status to be invoked on the listener.

Returns:

newly created publisher object or NULL on failure.

See also:

Specifying QoS on entities (p. 267) for information on setting QoS before entity creation
DDS::PublisherQos (p. 1074) for rules on consistency among QoS
DDS::DomainParticipant::PUBLISHER_QOS_DEFAULT (p. 38)
DDS::DomainParticipant::create_publisher_with_profile (p. 616)
DDS::DomainParticipant::get_default_publisher_qos (p. 610)
DDS::Publisher::set_listener (p. 1065)

Examples:

HelloWorld_publisher.cpp.

6.44.3.14 Publisher ^ DDS::DomainParticipant::create_publisher_with_profile (System::String^ *library_name*, System::String^ *profile_name*, PublisherListener^ *listener*, StatusMask *mask*)

<<eXtension>> *(p. 174)* Creates a new **DDS::Publisher** (p. 1044) object using the **DDS::PublisherQos** (p. 1074) associated with the input XML QoS profile.

Precondition:

The **DDS::PublisherQos** (p. 1074) in the input profile must be consistent, or the operation will fail and no **DDS::Publisher** (p. 1044) will be created.

Parameters:

library_name *<<in>>* *(p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipant::set_default_library** (p. 596)).

profile_name *<<in>>* *(p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipant::set_default_profile** (p. 597)).

listener *<<in>>* *(p. 175)*. **Listener** (p. 952) to be attached to the newly created **DDS::Publisher** (p. 1044).

mask *<<in>>* *(p. 175)*. Changes of communication status to be invoked on the listener.

Returns:

newly created publisher object or NULL on failure.

See also:

Specifying QoS on entities (p. 267) for information on setting QoS before entity creation
DDS::PublisherQos (p. 1074) for rules on consistency among QoS
DDS::DomainParticipant::create_publisher (p. 615)
DDS::DomainParticipant::get_default_publisher_qos (p. 610)
DDS::Publisher::set_listener (p. 1065)

6.44.3.15 void DDS::DomainParticipant::delete_publisher (Publisher^ % *p*)

Deletes an existing **DDS::Publisher** (p. 1044).

Precondition:

The **DDS::Publisher** (p. 1044) must not have any attached **DDS::DataWriter** (p. 499) objects. If there are existing **DDS::DataWriter** (p. 499) objects, it will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123).
DDS::Publisher (p. 1044) must have been created by this **DDS::DomainParticipant** (p. 577), or else it will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123).

Postcondition:

Listener (p. 952) installed on the **DDS::Publisher** (p. 1044) will not be called after this method completes successfully.

Parameters:

p *<<in>> (p. 175)* **DDS::Publisher** (p. 1044) to be deleted.

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_PreconditionNotMet** (p. 1123).

6.44.3.16 Subscriber ^ DDS::DomainParticipant::create_subscriber (SubscriberQos^ *qos*, SubscriberListener^ *listener*, StatusMask *mask*)

Creates a **DDS::Subscriber** (p. 1201) with the desired QoS policies and attaches to it the specified **DDS::SubscriberListener** (p. 1226).

Precondition:

The specified QoS policies must be consistent, or the operation will fail and no **DDS::Subscriber** (p. 1201) will be created.

MT Safety:

UNSAFE. If **DDS::DomainParticipant::SUBSCRIBER_QOS_DEFAULT** (p. 38) 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. 613).

Parameters:

qos *<<in>> (p. 175)* QoS to be used for creating the new **DDS::Subscriber** (p. 1201). The special value **DDS::DomainParticipant::SUBSCRIBER_QOS_DEFAULT** (p. 38) can be used to indicate that the **DDS::Subscriber** (p. 1201) should be created with the default **DDS::SubscriberQos** (p. 1230) set in the **DDS::DomainParticipant** (p. 577). Cannot be NULL.

listener *<<in>> (p. 175).* **Listener** (p. 952) to be attached to the newly created **DDS::Subscriber** (p. 1201).

mask *<<in>> (p. 175).* Changes of communication status to be invoked on the listener.

Returns:

newly created subscriber object or NULL on failure.

See also:

Specifying QoS on entities (p. 267) for information on setting QoS before entity creation
DDS::SubscriberQos (p. 1230) for rules on consistency among QoS
DDS::DomainParticipant::SUBSCRIBER_QOS_DEFAULT (p. 38)
DDS::DomainParticipant::create_subscriber_with_profile (p. 619)
DDS::DomainParticipant::get_default_subscriber_qos (p. 612)
DDS::Subscriber::set_listener (p. 1222)

Examples:

HelloWorld_subscriber.cpp.

6.44.3.17 Subscriber ^ DDS::DomainParticipant::create_subscriber_with_profile (System::String^ *library_name*, System::String^ *profile_name*, SubscriberListener^ *listener*, StatusMask *mask*)

<<eXtension>> (p. 174) Creates a new **DDS::Subscriber** (p. 1201) object using the **DDS::PublisherQos** (p. 1074) associated with the input XML QoS profile.

Precondition:

The **DDS::SubscriberQos** (p. 1230) in the input profile must be consistent, or the operation will fail and no **DDS::Subscriber** (p. 1201) will be created.

Parameters:

library_name *<<in>> (p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipant::set_default_library** (p. 596)).

profile_name *<<in>> (p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipant::set_default_profile** (p. 597)).

listener *<<in>> (p. 175).* **Listener** (p. 952) to be attached to the newly created **DDS::Subscriber** (p. 1201).

mask *<<in>> (p. 175).* Changes of communication status to be invoked on the listener.

Returns:

newly created subscriber object or NULL on failure.

See also:

Specifying QoS on entities (p. 267) for information on setting QoS before entity creation
DDS::SubscriberQos (p. 1230) for rules on consistency among QoS
DDS::DomainParticipant::create_subscriber (p. 618)
DDS::DomainParticipant::get_default_subscriber_qos (p. 612)
DDS::Subscriber::set_listener (p. 1222)

6.44.3.18 void DDS::DomainParticipant::delete_subscriber (Subscriber^ % *s*)

Deletes an existing **DDS::Subscriber** (p. 1201).

Precondition:

The **DDS::Subscriber** (p. 1201) must not have any attached **DDS::DataReader** (p. 433) objects. If there are existing **DDS::DataReader** (p. 433) objects, it will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123)
The **DDS::Subscriber** (p. 1201) must have been created by this **DDS::DomainParticipant** (p. 577), or else it will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123).

Postcondition:

A **Listener** (p. 952) installed on the **DDS::Subscriber** (p. 1201) will not be called after this method completes successfully.

Parameters:

s *<<in>>* *(p. 175)* **DDS::Subscriber** (p. 1201) to be deleted.

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_PreconditionNotMet** (p. 1123).

6.44.3.19 void DDS::DomainParticipant::get_publishers (PublisherSeq^ *publishers*)

<<eXtension>> *(p. 174)* 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_OutOfResources** (p. 1122).

MT Safety:

Safe.

Parameters:

publishers *<<**inout**>>* *(p. 176)* a **PublisherSeq** (p. 1076) object where the set or list of publishers will be returned

Returns:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_OutOfResources** (p. 1122)

6.44.3.20 void DDS::DomainParticipant::get_subscribers (SubscriberSeq^ *subscribers*)

*<<**eXtension**>>* *(p. 174)* 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_OutOfResources** (p. 1122).

MT Safety:

Safe.

Parameters:

subscribers *<<**inout**>>* *(p. 176)* a **SubscriberSeq** (p. 1232) object where the set or list of subscribers will be returned

Returns:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_OutOfResources** (p. 1122)

6.44.3.21 Topic ^ DDS::DomainParticipant::create_topic (System::String^ *topic_name*, System::String^ *type_name*, TopicQos^ *qos*, TopicListener^ *listener*, StatusMask *mask*)

Creates a **DDS::Topic** (p. 1258) with the desired QoS policies and attaches to it the specified **DDS::TopicListener** (p. 1278).

Precondition:

The application is not allowed to create two **DDS::Topic** (p. 1258) objects with the same `topic_name` attached to the same **DDS::DomainParticipant** (p. 577). If the application attempts this, this method will fail and return a NULL topic.
The specified QoS policies must be consistent, or the operation will fail and no **DDS::Topic** (p. 1258) will be created.
Prior to creating a **DDS::Topic** (p. 1258), the type must have been registered with RTI Data Distribution Service. This is done using the **FooTypeSupport::register_type** (p. 885) operation on a derived class of the **DDS::TypeSupport** (p. 1385) 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. 629).

MT Safety:

UNSAFE. If **DDS::DomainParticipant::TOPIC_QOS_DEFAULT** (p. 39) 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. 608).

Parameters:

topic_name *<<in>> (p. 175)* Name for the new topic, must not exceed 255 characters. Cannot be NULL.

type_name *<<in>> (p. 175)* The type to which the new **DDS::Topic** (p. 1258) will be bound. Cannot be NULL.

qos *<<in>> (p. 175)* QoS to be used for creating the new **DDS::Topic** (p. 1258). The special value **DDS::DomainParticipant::TOPIC_QOS_DEFAULT** (p. 39) can be used to indicate that the **DDS::Topic** (p. 1258) should be created with the default **DDS::TopicQos** (p. 1280) set in the **DDS::DomainParticipant** (p. 577). Cannot be NULL.

listener *<<in>> (p. 175)*. **Listener** (p. 952) to be attached to the newly created **DDS::Topic** (p. 1258).

mask *<<in>> (p. 175)*. Changes of communication status to be invoked on the listener.

Returns:

newly created topic, or NULL on failure

See also:

Specifying QoS on entities (p. 267) for information on setting QoS before entity creation
DDS::TopicQos (p. 1280) for rules on consistency among QoS
DDS::DomainParticipant::TOPIC_QOS_DEFAULT (p. 39)
DDS::DomainParticipant::create_topic_with_profile (p. 623)
DDS::DomainParticipant::get_default_topic_qos (p. 607)
DDS::Topic::set_listener (p. 1263)

Examples:

HelloWorld_publisher.cpp, and **HelloWorld_subscriber.cpp**.

6.44.3.22 Topic ^ DDS::DomainParticipant::create_topic_with_profile (System::String^ *topic_name*, System::String^ *type_name*, System::String^ *library_name*, System::String^ *profile_name*, TopicListener^ *listener*, StatusMask *mask*)

<<eXtension>> (p. 174) Creates a new **DDS::Topic** (p. 1258) object using the **DDS::PublisherQos** (p. 1074) associated with the input XML QoS profile.

Precondition:

The application is not allowed to create two DDS::TopicDescription objects with the same `topic_name` attached to the same **DDS::DomainParticipant** (p. 577). If the application attempts this, this method will fail and return a NULL topic.
The **DDS::TopicQos** (p. 1280) in the input profile must be consistent, or the operation will fail and no **DDS::Topic** (p. 1258) will be created.
Prior to creating a **DDS::Topic** (p. 1258), the type must have been registered with RTI Data Distribution Service. This is done using the **FooTypeSupport::register_type** (p. 885) operation on a derived class of the **DDS::TypeSupport** (p. 1385) 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. 629).

Parameters:

topic_name *<<in>> (p. 175)* Name for the new topic, must not exceed 255 characters. Cannot be NULL.

type_name *<<in>>* *(p. 175)* The type to which the new **DDS::Topic** (p. 1258) will be bound. Cannot be NULL.

library_name *<<in>>* *(p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipant::set_default_library** (p. 596)).

profile_name *<<in>>* *(p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipant::set_default_profile** (p. 597)).

listener *<<in>>* *(p. 175)*. **Listener** (p. 952) to be attached to the newly created **DDS::Topic** (p. 1258).

mask *<<in>>* *(p. 175)*. Changes of communication status to be invoked on the listener.

Returns:

newly created topic, or NULL on failure

See also:

Specifying QoS on entities (p. 267) for information on setting QoS before entity creation
DDS::TopicQos (p. 1280) for rules on consistency among QoS
DDS::DomainParticipant::create_topic (p. 621)
DDS::DomainParticipant::get_default_topic_qos (p. 607)
DDS::Topic::set_listener (p. 1263)

6.44.3.23 void DDS::DomainParticipant::delete_topic (Topic^ % *topic*)

Deletes a **DDS::Topic** (p. 1258).

Precondition:

If the **DDS::Topic** (p. 1258) does not belong to the application's **DDS::DomainParticipant** (p. 577), this operation fails with **DDS::Retcode_PreconditionNotMet** (p. 1123).
Make sure no objects are using the topic. More specifically, there must be no existing **DDS::DataReader** (p. 433), **DDS::DataWriter** (p. 499), **DDS::ContentFilteredTopic** (p. 419), or **DDS::MultiTopic** (p. 984) objects belonging to the same **DDS::DomainParticipant** (p. 577) that are using the **DDS::Topic** (p. 1258). If delete_topic is called on a **DDS::Topic** (p. 1258) with any of these existing objects attached to it, it will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123).

Postcondition:

Listener (p. 952) installed on the **DDS::Topic** (p. 1258) will not be called after this method completes successfully.

Parameters:

topic *<<in>>* *(p. 175)* **DDS::Topic** (p. 1258) to be deleted.

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_PreconditionNotMet** (p. 1123)

6.44.3.24 ContentFilteredTopic ^ DDS::DomainParticipant::create_contentfilteredtopic (System::String^ *name*, Topic^ *related_topic*, System::String^ *filter_expression*, StringSeq^ *expression_parameters*)

Creates a **DDS::ContentFilteredTopic** (p. 419), that can be used to do content-based subscriptions.

The **DDS::ContentFilteredTopic** (p. 419) only relates to samples published under that **DDS::Topic** (p. 1258), filtered according to their content. The filtering is done by means of evaluating a logical expression that involves the values of some of the data-fields in the sample. The logical expression derived from the `filter_expression` and `expression_parameters` arguments.

Queries and Filters Syntax (p. 184) describes the syntax of `filter_expression` and `expression_parameters`.

Precondition:

The application is not allowed to create two **DDS::ContentFilteredTopic** (p. 419) objects with the same `topic_name` attached to the same **DDS::DomainParticipant** (p. 577). If the application attempts this, this method will fail and returns NULL.
If `related_topic` does not belong to this **DDS::DomainParticipant** (p. 577), 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. Do not use rtiddsgen's -notypecode option if you want to use the builtin SQL filter.

Parameters:

name *<<in>>* *(p. 175)* Name for the new content filtered topic, must not exceed 255 characters. Cannot be NULL.

related_topic *<<in>>* *(p. 175)* **DDS::Topic** (p. 1258) to be filtered. Cannot be NULL.

filter_expression *<<in>>* *(p. 175)* Cannot be NULL

expression_parameters *<<in>>* *(p. 175)* 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. 419), or NULL on failure

6.44.3.25 ContentFilteredTopic ^ DDS::DomainParticipant::create_contentfilteredtopic_with_filter (System::String^ *name*, Topic^ *related_topic*, System::String^ *filter_expression*, StringSeq^ *expression_parameters*, System::String^ *filter_name*)

<<eXtension>> *(p. 174)* Creates a **DDS::ContentFilteredTopic** (p. 419) using the specified filter to do content-based subscriptions.

Parameters:

name *<<in>>* *(p. 175)* Name for the new content filtered topic. Cannot exceed 255 characters. Cannot be NULL.

related_topic *<<in>>* *(p. 175)* **DDS::Topic** (p. 1258) to be filtered. Cannot be NULL.

filter_expression *<<in>>* *(p. 175)* Cannot be NULL.

expression_parameters *<<in>>* *(p. 175)* 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.

filter_name *<<in>>* *(p. 175)* Name of content filter to use. Must previously have been registered with **DDS::DomainParticipant::register_contentfilter** (p. 593) on the same **DDS::DomainParticipant** (p. 577). Cannot be NULL.

Builtin filter names are **DDS::DomainParticipant::SQLFILTER_NAME** (p. 40) and **DDS::DomainParticipant::STRINGMATCHFILTER_NAME** (p. 41)

Returns:

newly created **DDS::ContentFilteredTopic** (p. 419), or NULL on failure

6.44.3.26 void DDS::DomainParticipant::delete_contentfilteredtopic (ContentFilteredTopic^ % *a_contentfilteredtopic*)

Deletes a **DDS::ContentFilteredTopic** (p. 419).

Precondition:

The deletion of a **DDS::ContentFilteredTopic** (p. 419) is not allowed if there are any existing **DDS::DataReader** (p. 433) objects that are using the **DDS::ContentFilteredTopic** (p. 419). If the operation is called on a **DDS::ContentFilteredTopic** (p. 419) with existing **DDS::DataReader** (p. 433) objects attached to it, it will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123).
The **DDS::ContentFilteredTopic** (p. 419) must be created by this **DDS::DomainParticipant** (p. 577), or else this operation will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123).

Parameters:

a_contentfilteredtopic *<<in>> (p. 175)*

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_PreconditionNotMet** (p. 1123)

6.44.3.27 MultiTopic ^ DDS::DomainParticipant::create_multitopic (System::String^ *name*, System::String^ *type_name*, System::String^ *subscription_expression*, StringSeq^ *expression_parameters*) [inline]

[**Not supported (optional)**] Creates a **MultiTopic** (p. 984) 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. 1258) are specified using the `subscription_expression` and `expression_parameters` arguments.

Queries and Filters Syntax (p. 184) describes the syntax of `subscription_expression` and `expression_parameters`.

Precondition:

The application is not allowed to create two DDS::TopicDescription objects with the same `name` attached to the same **DDS::DomainParticipant** (p. 577). If the application attempts this, this method will fail and return NULL.
Prior to creating a **DDS::MultiTopic** (p. 984), the type must have been registered with RTI Data Distribution Service. This is done using the **FooTypeSupport::register_type** (p. 885) operation on a derived class of the **DDS::TypeSupport** (p. 1385) interface. Otherwise, this method will return NULL.

Parameters:

name *<<in>>* *(p. 175)* Name of the newly create **DDS::MultiTopic** (p. 984). Cannot be NULL.

type_name *<<in>>* *(p. 175)* Cannot be NULL.

subscription_expression *<<in>>* *(p. 175)* Cannot be NULL.

expression_parameters *<<in>>* *(p. 175)* Cannot be NULL.

Returns:

NULL

6.44.3.28 void DDS::DomainParticipant::delete_multitopic (MultiTopic^ *a_multitopic*) `[inline]`

[**Not supported (optional)**] Deletes a **DDS::MultiTopic** (p. 984).

Precondition:

The deletion of a **DDS::MultiTopic** (p. 984) is not allowed if there are any existing **DDS::DataReader** (p. 433) objects that are using the **DDS::MultiTopic** (p. 984). If the delete_multitopic operation is called on a **DDS::MultiTopic** (p. 984) with existing **DDS::DataReader** (p. 433) objects attached to it, it will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123).
The **DDS::MultiTopic** (p. 984) must be created by this **DDS::DomainParticipant** (p. 577), or else this operation will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123).

Parameters:

a_multitopic *<<in>>* *(p. 175)*

Exceptions:

DDS::Retcode_Unsupported *(p. 1125)*

6.44.3.29 Topic ^ DDS::DomainParticipant::find_topic (System::String^ *topic_name*, Duration_t% *timeout*)

Finds an existing (or ready to exist) **DDS::Topic** (p. 1258), based on its name.

This call can be used to block for a specified duration to wait for the **DDS::Topic** (p. 1258) to be created.

If the requested **DDS::Topic** (p. 1258) already exists, it is returned. Otherwise, **find_topic()** (p. 629) waits until another thread creates it or else returns when the specified timeout occurs.

find_topic() (p. 629) is useful when multiple threads are concurrently creating and looking up topics. In that case, one thread can call **find_topic()** (p. 629) 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. 629).

The **DDS::DomainParticipant** (p. 577) must already be enabled.

Note: *Each* **DDS::Topic** (p. 1258) obtained by **DDS::DomainParticipant::find_topic** (p. 629) must also be deleted by means of **DDS::DomainParticipant::delete_topic** (p. 624). If **DDS::Topic** (p. 1258) is obtained multiple times by means of **DDS::DomainParticipant::find_topic** (p. 629) or **DDS::DomainParticipant::create_topic** (p. 621), it must also be deleted that same number of times using **DDS::DomainParticipant::delete_topic** (p. 624).

Parameters:

topic_name *<<in>> (p. 175)* Name of the **DDS::Topic** (p. 1258) to search for. Cannot be NULL.

timeout *<<in>> (p. 175)* The time to wait if the **DDS::Topic** (p. 1258) does not exist already. Cannot be NULL.

Returns:

the topic, if it exists, or NULL

6.44.3.30 ITopicDescription ^ DDS::DomainParticipant::lookup_topicdescription (System::String^ *topic_name*)

Looks up an existing, locally created DDS::TopicDescription, based on its name.

DDS::TopicDescription is the base class for **DDS::Topic** (p. 1258), **DDS::MultiTopic** (p. 984) and **DDS::ContentFilteredTopic** (p. 419).

So you can narrow the DDS::TopicDescription returned from this operation to a **DDS::Topic** (p. 1258) or **DDS::ContentFilteredTopic** (p. 419) as appropriate.

Unlike **DDS::DomainParticipant::find_topic** (p. 629), which logically returns a new **DDS::Topic** (p. 1258) object that must be independently deleted, *this* operation returns a reference to the original local object.

The **DDS::DomainParticipant** (p. 577) does not have to be enabled when you call **lookup_topicdescription()** (p. 629).

The returned topic may be either enabled or disabled.

MT Safety:

UNSAFE. It is not safe to lookup a topic description while another thread is creating that topic.

Parameters:

topic_name *<<in>> (p. 175)* Name of DDS::TopicDescription 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.

6.44.3.31 FlowController ^ DDS::DomainParticipant::create_flowcontroller (System::String^ *name*, FlowControllerProperty_t^ *prop*)

<<eXtension>> (p. 174) Creates a **DDS::FlowController** (p. 867) with the desired property.

The created **DDS::FlowController** (p. 867) is associated with a **DDS::DataWriter** (p. 499) via **DDS::PublishModeQosPolicy::flow_controller_name** (p. 1079). A single **DDS::FlowController** (p. 867) may service multiple **DDS::DataWriter** (p. 499) instances, even if they belong to a different **DDS::Publisher** (p. 1044). The `property` determines how the **DDS::FlowController** (p. 867) shapes the network traffic.

Precondition:

The specified `property` must be consistent, or the operation will fail and no **DDS::FlowController** (p. 867) will be created.

MT Safety:

UNSAFE. If DDS::FLOW_CONTROLLER_PROPERTY_DEFAULT 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. 592) or trying to lookup that flow controller with **DDS::DomainParticipant::lookup_flowcontroller** (p. 632).

Parameters:

name *<<**in**>>* *(p. 175)* name of the **DDS::FlowController** (p. 867) to create. A **DDS::DataWriter** (p. 499) is associated with a **DDS::FlowController** (p. 867) by name. Limited to 255 characters.

prop *<<**in**>>* *(p. 175)* property to be used for creating the new **DDS::FlowController** (p. 867). The special value DDS::FLOW_CONTROLLER_PROPERTY_DEFAULT can be used to indicate that the **DDS::FlowController** (p. 867) should be created with the default **DDS::FlowControllerProperty_t** (p. 871) set in the **DDS::DomainParticipant** (p. 577). Cannot be NULL.

Returns:

Newly created flow controller object or NULL on failure.

See also:

DDS::FlowControllerProperty_t (p. 871) for rules on consistency among property
DDS::FLOW_CONTROLLER_PROPERTY_DEFAULT
DDS::DomainParticipant::get_default_flowcontroller_property (p. 592)

6.44.3.32 void DDS::DomainParticipant::delete_flowcontroller (FlowController^ % *fc*)

*<<**eXtension**>>* *(p. 174)* Deletes an existing **DDS::FlowController** (p. 867).

Precondition:

The **DDS::FlowController** (p. 867) must not have any attached **DDS::DataWriter** (p. 499) objects. If there are any attached **DDS::DataWriter** (p. 499) objects, it will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123).
The **DDS::FlowController** (p. 867) must have been created by this **DDS::DomainParticipant** (p. 577), or else it will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123).

Postcondition:

The **DDS::FlowController** (p. 867) is deleted if this method completes successfully.

Parameters:

fc *<<in>> (p. 175)* The **DDS::FlowController** (p. 867) to be deleted.

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_PreconditionNotMet** (p. 1123).

6.44.3.33 FlowController ^ DDS::DomainParticipant::lookup_flowcontroller (System::String^ *name*)

<<eXtension>> (p. 174) Looks up an existing locally-created **DDS::FlowController** (p. 867), based on its name.

Looks up a previously created **DDS::FlowController** (p. 867), including the built-in ones. Once a **DDS::FlowController** (p. 867) has been deleted, subsequent lookups will fail.

MT Safety:

UNSAFE. It is not safe to lookup a flow controller description while another thread is creating that flow controller.

Parameters:

name *<<in>> (p. 175)* Name of **DDS::FlowController** (p. 867) to search for. Limited to 255 characters. Cannot be NULL.

Returns:

The flow controller if it has already been created locally, or NULL otherwise.

6.44.3.34 Subscriber ^ DDS::DomainParticipant::get_builtin_subscriber ()

Accesses the **built-in DDS::Subscriber** (p. 1201).

Each **DDS::DomainParticipant** (p. 577) contains several built-in **DDS::Topic** (p. 1258) objects as well as corresponding **DDS::DataReader** (p. 433) objects to access them. All of these **DDS::DataReader** (p. 433) objects belong to a single built-in **DDS::Subscriber** (p. 1201).

The built-in Topics are used to communicate information about other **DDS::DomainParticipant** (p. 577), **DDS::Topic** (p. 1258), **DDS::DataReader** (p. 433), and **DDS::DataWriter** (p. 499) 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. 577) is deleted.

Returns:

The built-in **DDS::Subscriber** (p. 1201) singleton.

See also:

DDS::SubscriptionBuiltinTopicData (p. 1233)
DDS::PublicationBuiltinTopicData (p. 1030)
DDS::ParticipantBuiltinTopicData (p. 1002)
DDS::TopicBuiltinTopicData (p. 1268)

6.44.3.35 void DDS::DomainParticipant::ignore_participant (InstanceHandle_t% *handle*)

Instructs RTI Data Distribution Service to locally ignore a remote **DDS::DomainParticipant** (p. 577).

From the time of this call onwards, RTI Data Distribution Service 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. 577).

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. 1002) to provide access control.

Application data can be associated with a **DDS::DomainParticipant** (p. 577) by means of the **USER_DATA** (p. 273) 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. 577) to ignore is identified by the `handle` argument. This `handle` is the one that appears in the **DDS::SampleInfo** (p. 1148) retrieved when reading the data-samples available for the built-in **DDS::DataReader** (p. 433) to the `DDS::DomainParticipant` (p. 577) topic. The built-in **DDS::DataReader** (p. 433) is read with the same **DDS::TypedDataReader::read** (p. 1341) and **DDS::TypedDataReader::take** (p. 1342) operations used for any **DDS::DataReader** (p. 433).

Parameters:

handle *<<in>>* *(p. 175)* **DDS::InstanceHandle_t** (p. 905) of the **DDS::DomainParticipant** (p. 577) to be ignored. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_OutOfResources** (p. 1122), **DDS::Retcode_NotEnabled** (p. 1121)

See also:

DDS::ParticipantBuiltinTopicData (p. 1002)
DDS::ParticipantBuiltinTopicDataTypeSupport::PARTICIPANT_TOPIC_NAME (p. 226)
DDS::DomainParticipant::get_builtin_subscriber (p. 632)

6.44.3.36 void DDS::DomainParticipant::ignore_topic (InstanceHandle_t% *handle*)

Instructs RTI Data Distribution Service to locally ignore a **DDS::Topic** (p. 1258).

This means it will locally ignore any publication, or subscription to the **DDS::Topic** (p. 1258).

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. 1258) to ignore is identified by the `handle` argument. This is the handle of a **DDS::Topic** (p. 1258) that appears in the **DDS::SampleInfo** (p. 1148) retrieved when reading data samples from the built-in **DDS::DataReader** (p. 433) for the **DDS::Topic** (p. 1258).

Parameters:

handle *<<in>>* *(p. 175)* Handle of the **DDS::Topic** (p. 1258) to be ignored. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_OutOfResources** (p. 1122) or **DDS::Retcode_NotEnabled** (p. 1121)

See also:

DDS::TopicBuiltinTopicData (p. 1268)

DDS::TopicBuiltinTopicDataTypeSupport::TOPIC_TOPIC_NAME (p. 228)
DDS::DomainParticipant::get_builtin_subscriber (p. 632)

6.44.3.37 void DDS::DomainParticipant::ignore_publication (InstanceHandle_t% *handle*)

Instructs RTI Data Distribution Service 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. 1044) (see **DDS::PublicationBuiltinTopicData** (p. 1030)). After this call, any data written by that publication's **DDS::DataWriter** (p. 499) will be ignored.

This operation can be used to ignore local *and* remote DataWriters.

The publication (**DataWriter** (p. 499)) to ignore is identified by the `handle` argument.

- To ignore a *remote* **DataWriter** (p. 499), the `handle` can be obtained from the **DDS::SampleInfo** (p. 1148) retrieved when reading data samples from the built-in **DDS::DataReader** (p. 433) for the publication topic.

- To ignore a *local* **DataWriter** (p. 499), the `handle` can be obtained by calling **DDS::Entity::get_instance_handle** (p. 850) for the local **DataWriter** (p. 499).

There is no way to reverse this operation.

Parameters:

handle *<<in>> (p. 175)* Handle of the **DDS::DataWriter** (p. 499) to be ignored. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_OutOfResources** (p. 1122) or **DDS::Retcode_NotEnabled** (p. 1121)

See also:

DDS::PublicationBuiltinTopicData (p. 1030)
DDS::PublicationBuiltinTopicDataTypeSupport::PUBLICATION_TOPIC_NAME (p. 230)
DDS::DomainParticipant::get_builtin_subscriber (p. 632)

6.44.3.38 void DDS::DomainParticipant::ignore_subscription (InstanceHandle_t% *handle*)

Instructs RTI Data Distribution Service 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. 1201) (see **DDS::SubscriptionBuiltinTopicData** (p. 1233)). After this call, any data received related to that subscription's **DDS::DataReader** (p. 433) 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** (p. 433), the `handle` can be obtained from the **DDS::SampleInfo** (p. 1148) retrieved when reading data samples from the built-in **DDS::DataReader** (p. 433) for the subscription topic.

- To ignore a *local* **DataReader** (p. 433), the `handle` can be obtained by calling **DDS::Entity::get_instance_handle** (p. 850) for the local **DataReader** (p. 433).

There is no way to reverse this operation.

Parameters:

handle *<<in>> (p. 175)* Handle of the **DDS::DataReader** (p. 433) to be ignored. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_OutOfResources** (p. 1122) or **DDS::Retcode_NotEnabled** (p. 1121)

See also:

DDS::SubscriptionBuiltinTopicData (p. 1233)
DDS::SubscriptionBuiltinTopicDataTypeSupport::SUBSCRIPTION_TOPIC_NAME (p. 232)
DDS::DomainParticipant::get_builtin_subscriber (p. 632)

6.44.3.39 System::Int32 DDS::DomainParticipant::get_domain_id ()

Get the unique domain identifier.

This operation retrieves the `domain_id` used to create the **DDS::DomainParticipant** (p. 577). The `domain_id` identifies the DDS domain to which the **DDS::DomainParticipant** (p. 577) belongs. Each DDS domain represents a separate data 'communication plane' isolated from other domains.

Returns:

the unique `domainId` that was used to create the domain

See also:

DDS::DomainParticipantFactory::create_participant (p. 665)
DDS::DomainParticipantFactory::create_participant_with_profile (p. 667)

6.44.3.40 void DDS::DomainParticipant::get_current_time (Time_t% *current_time*)

Returns the current value of the time.

The current value of the time that RTI Data Distribution Service uses to time-stamp **DDS::DataWriter** (p. 499) and to set the reception-timestamp for the data updates that it receives.

Parameters:

current_time <<*inout*>> *(p. 176)* Current time to be filled up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.44.3.41 void DDS::DomainParticipant::assert_liveliness ()

Manually asserts the liveliness of this **DDS::DomainParticipant** (p. 577).

This is used in combination with the **DDS::LivelinessQosPolicy** (p. 960) to indicate to RTI Data Distribution Service that the entity remains active.

You need to use this operation if the **DDS::DomainParticipant** (p. 577) conatins **DDS::DataWriter** (p. 499) entities with the **DDS::LivelinessQosPolicy::kind** (p. 963) set to DDS::LivelinessQosPolicyKind::MANUAL_BY_PARTICIPANT_-LIVELINESS_QOS and it only affects the liveliness of those **DDS::DataWriter** (p. 499) entities. Otherwise, it has no effect.

Note: writing data via the **DDS::TypedDataWriter::write** (p. 1376) or **DDS::TypedDataWriter::write_w_timestamp** (p. 1378) operation asserts liveliness on the **DDS::DataWriter** (p. 499) itself and its **DDS::DomainParticipant** (p. 577). Consequently the use of **assert_liveliness()** (p. 637) is only needed if the application is not writing data regularly.

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_NotEnabled** (p. 1121)

See also:

DDS::LivelinessQosPolicy (p. 960)

6.44.3.42 void DDS::DomainParticipant::delete_contained_entities ()

Delete all the entities that were created by means of the "create" operations on the **DDS::DomainParticipant** (p. 577).

This operation deletes all contained **DDS::Publisher** (p. 1044) (including an implicit **Publisher** (p. 1044), if one exists), **DDS::Subscriber** (p. 1201) (including implicit subscriber), **DDS::Topic** (p. 1258), **DDS::ContentFilteredTopic** (p. 419), and **DDS::MultiTopic** (p. 984) 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()** (p. 638) on the **DDS::DomainParticipant** (p. 577) will end up deleting all the entities recursively contained in the **DDS::DomainParticipant** (p. 577), that is also the **DDS::DataWriter** (p. 499), **DDS::DataReader** (p. 433), as well as the **DDS::QueryCondition** (p. 1082) and **DDS::ReadCondition** (p. 1084) objects belonging to the contained **DDS::DataReader** (p. 433).

The operation will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123) if any of the contained entities is in a state where it cannot be deleted.

If **delete_contained_entities()** (p. 638) completes successfully, the application may delete the **DDS::DomainParticipant** (p. 577) knowing that it has no contained entities.

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_PreconditionNotMet** (p. 1123).

Examples:

HelloWorld_publisher.cpp, and **HelloWorld_subscriber.cpp**.

6.44.3.43 void DDS::DomainParticipant::get_discovered_participants (InstanceHandleSeq^ *participant_handles*)

Returns list of discovered **DDS::DomainParticipant** (p. 577) s.

This operation retrieves the list of **DDS::DomainParticipant** (p. 577) 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_participant** (p. 633) operation.

Parameters:

participant_handles <<*inout*>> *(p. 176)* **DDS::InstanceHandleSeq** (p. 906) to be filled with handles of the discovered **DDS::DomainParticipant** (p. 577) s

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_NotEnabled** (p. 1121)

6.44.3.44 void DDS::DomainParticipant::get_discovered_participant_data (ParticipantBuiltinTopicData^ *participant_data*, InstanceHandle_t% *participant_handle*)

Returns **DDS::ParticipantBuiltinTopicData** (p. 1002) for the specified **DDS::DomainParticipant** (p. 577) .

This operation retrieves information on a **DDS::DomainParticipant** (p. 577) 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. 633) operation.

The `participant_handle` must correspond to such a **DomainParticipant** (p. 577). Otherwise, the operation will fail with PRECONDITION_NOT_MET.

Use the operation **DDS::DomainParticipant::get_discovered_participants** (p. 639) to find the **DDS::DomainParticipant** (p. 577) s that are currently discovered.

Note: This operation does not retrieve the DDS::ParticipantBuiltinTopicData::property. This information is available through **DDS::DataReaderListener::on_data_available()** (p. 463) (if a reader listener is installed on the **DDS::ParticipantBuiltinTopicDataDataReader** (p. 1005)).

Parameters:

participant_data *<<inout>>* *(p. 176)* **DDS::ParticipantBuiltinTopicData** (p. 1002) to be filled with the specified **DDS::DomainParticipant** (p. 577) 's data.

participant_handle *<<in>>* *(p. 175)* **DDS::InstanceHandle_t** (p. 905) of **DDS::DomainParticipant** (p. 577).

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_PreconditionNotMet** (p. 1123) or **DDS::Retcode_NotEnabled** (p. 1121)

See also:

DDS::ParticipantBuiltinTopicData (p. 1002)
DDS::DomainParticipant::get_discovered_participants (p. 639)

6.44.3.45 void DDS::DomainParticipant::get_discovered_topics (InstanceHandleSeq^ *topic_handles*)

Returns list of discovered **DDS::Topic** (p. 1258) objects.

This operation retrieves the list of **DDS::Topic** (p. 1258) 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. 634) operation.

Parameters:

topic_handles *<<inout>>* *(p. 176)* **DDS::InstanceHandleSeq** (p. 906) to be filled with handles of the discovered **DDS::Topic** (p. 1258) objects

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_NotEnabled** (p. 1121)

6.44.3.46 void DDS::DomainParticipant::get_discovered_topic_data (TopicBuiltinTopicData^ *topic_data*, InstanceHandle_t% *topic_handle*)

Returns **DDS::TopicBuiltinTopicData** (p. 1268) for the specified **DDS::Topic** (p. 1258).

This operation retrieves information on a **DDS::Topic** (p. 1258) that has been discovered on the network. The topic must have been created by a participant 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_topic** (p. 634) operation.

The `topic_handle` must correspond to such a topic. Otherwise, the operation will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123).

This call is not supported for remote topics. If a remote `topic_handle` is used, the operation will fail with **DDS::Retcode_Unsupported** (p. 1125).

Use the operation **DDS::DomainParticipant::get_discovered_topics** (p. 640) to find the topics that are currently discovered.

Parameters:

topic_data <<*inout*>> *(p. 176)* **DDS::TopicBuiltinTopicData** (p. 1268) to be filled with the specified **DDS::Topic** (p. 1258)'s data.

topic_handle <<*in*>> *(p. 175)* **DDS::InstanceHandle_t** (p. 905) of **DDS::Topic** (p. 1258).

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_PreconditionNotMet** (p. 1123) or **DDS::Retcode_NotEnabled** (p. 1121)

See also:

DDS::TopicBuiltinTopicData (p. 1268)
DDS::DomainParticipant::get_discovered_topics (p. 640)

6.44.3.47 System::Boolean DDS::DomainParticipant::contains_entity (InstanceHandle_t% *a_handle*)

Completes successfully with true if the referenced **DDS::Entity** (p. 845) is contained by the **DDS::DomainParticipant** (p. 577).

This operation checks whether or not the given `a_handle` represents an **DDS::Entity** (p. 845) that was created from the **DDS::DomainParticipant**

(p. 577). The containment applies recursively. That is, it applies both to entities (DDS::TopicDescription, **DDS::Publisher** (p. 1044), or **DDS::Subscriber** (p. 1201)) created directly using the **DDS::DomainParticipant** (p. 577) as well as entities created using a contained **DDS::Publisher** (p. 1044), or **DDS::Subscriber** (p. 1201) as the factory, and so forth.

The `instance` handle for an **DDS::Entity** (p. 845) may be obtained from built-in topic data, from various statuses, or from the operation **DDS::Entity::get_instance_handle** (p. 850).

Parameters:

a_handle *<<in>>* *(p. 175)* **DDS::InstanceHandle_t** (p. 905) of the **DDS::Entity** (p. 845) to be checked.

Returns:

true if **DDS::Entity** (p. 845) is contained by the **DDS::DomainParticipant** (p. 577), or false otherwise.

6.44.3.48 void DDS::DomainParticipant::set_qos (DomainParticipantQos^ *qos*)

Change the QoS of this domain participant.

The **DDS::DomainParticipantQos::user_data** (p. 685) and **DDS::DomainParticipantQos::entity_factory** (p. 685) can be changed. The other policies are immutable.

Parameters:

qos *<<in>>* *(p. 175)* Set of policies to be applied to **DDS::DomainParticipant** (p. 577). Policies must be consistent. Immutable policies cannot be changed after **DDS::DomainParticipant** (p. 577) is enabled. The special value **DDS::DomainParticipantFactory::PARTICIPANT_QOS_DEFAULT** (p. 35) can be used to indicate that the QoS of the **DDS::DomainParticipant** (p. 577) should be changed to match the current default **DDS::DomainParticipantQos** (p. 683) set in the **DDS::DomainParticipantFactory** (p. 649). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_ImmutablePolicy** (p. 1118) if immutable policy is changed, or **DDS::Retcode_InconsistentPolicy** (p. 1119) if policies are inconsistent

See also:

DDS::DomainParticipantQos (p. 683) for rules on consistency among QoS
set_qos (abstract) (p. 846)

6.44.3.49 void DDS::DomainParticipant::set_qos_with_profile (System::String^ *library_name*, System::String^ *profile_name*)

<<eXtension>> (p. 174) Change the QoS of this domain participant using the input XML QoS profile.

The **DDS::DomainParticipantQos::user_data** (p. 685) and **DDS::DomainParticipantQos::entity_factory** (p. 685) can be changed. The other policies are immutable.

Parameters:

library_name *<<in>> (p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipantFactory::set_default_library** (p. 657)).

profile_name *<<in>> (p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipantFactory::set_default_profile** (p. 658)).

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_ImmutablePolicy** (p. 1118) if immutable policy is changed, or **DDS::Retcode_InconsistentPolicy** (p. 1119) if policies are inconsistent

See also:

DDS::DomainParticipantQos (p. 683) for rules on consistency among QoS

6.44.3.50 void DDS::DomainParticipant::get_qos (DomainParticipantQos^ *qos*)

Get the participant QoS.

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

Parameters:

qos *<<**inout**>>* *(p. 176)* QoS to be filled up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

get_qos (abstract) (p. 847)

6.44.3.51 void DDS::DomainParticipant::add_peer (System::String^ *peer_desc_string*)

*<<**eXtension**>>* *(p. 174)* 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. 577) will try to communicate.

This method may be called at any time after this **DDS::DomainParticipant** (p. 577) has been created, before or after it has been enabled. If it is called after **DDS::Entity::enable** (p. 848), an attempt will be made to contact the new peer(s) immediately. If it is called before, the peer description will simply be added to the list that was populated by **DDS::DiscoveryQosPolicy::initial_peers** (p. 573); the first attempted contact will take place after this **DDS::DomainParticipant** (p. 577) is enabled.

Adding a peer description with this method does not guarantee that any peer(s) discovered as a result will exactly correspond to those described:

- This **DDS::DomainParticipant** (p. 577) will attempt to discover peer participants at the given locations but may not succeed if no such participants are available. Such a situation will not result in an error result from this method, which will not wait for contact attempt(s) to be made.
- If remote participants such as are described by the given peer description are discovered, the distributed application is configured with asymmetric peer lists, and **DDS::DiscoveryQosPolicy::accept_unknown_peers** (p. 574) is set to true, this **DDS::DomainParticipant** (p. 577) may actually discover *more* peers than are described in the given peer description.

To be informed of the exact remote participants that are discovered, regardless of which peers this **DDS::DomainParticipant** (p. 577) *attempts* to discover, use the built-in participant topic:

DDS::ParticipantBuiltinTopicDataTypeSupport::PARTICIPANT_-TOPIC_NAME (p. 226).

Note that there is no "remove peer" operation. To cease communications with a peer **DDS::DomainParticipant** (p. 577) that has been discovered, use **DDS::DomainParticipant::ignore_participant** (p. 633).

Adding a peer description with this method has no effect on the **DDS::DiscoveryQosPolicy::initial_peers** (p. 573) that may be subsequently retrieved with **DDS::DomainParticipant::get_qos()** (p. 643) (because **DDS::DiscoveryQosPolicy** (p. 571) is immutable).

Parameters:

peer_desc_string *<<in>> (p. 175)* New peer descriptor to be added. The format is specified in **Peer Descriptor Format** (p. 313). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

Peer Descriptor Format (p. 313)
DDS::DiscoveryQosPolicy::initial_peers (p. 573)
DDS::ParticipantBuiltinTopicDataTypeSupport::PARTICIPANT_-TOPIC_NAME (p. 226)
DDS::DomainParticipant::get_builtin_subscriber (p. 632)

6.44.3.52 void DDS::DomainParticipant::set_listener (DomainParticipantListener^ *l*, StatusMask *mask*)

Sets the participant listener.

Parameters:

l *<<in>> (p. 175)* **Listener** (p. 952) to be installed on entity.

mask *<<in>> (p. 175)* Changes of communication status to be invoked on the listener.

MT Safety:

Unsafe. This method is not synchronized with the listener callbacks, so it is possible to set a new listener on a participant when the old listener is in a callback.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

set_listener (abstract) (p. 847)

6.44.3.53 DomainParticipantListener ^ DDS::DomainParticipant::get_listener ()

Get the participant listener.

Returns:

Existing listener attached to the **DDS::DomainParticipant** (p. 577).

See also:

get_listener (abstract) (p. 848)

6.44.3.54 virtual void DDS::DomainParticipant::enable () [override, virtual]

Enables the **DDS::Entity** (p. 845).

This operation enables the **Entity** (p. 845). **Entity** (p. 845) objects can be created either enabled or disabled. This is controlled by the value of the **ENTITY_FACTORY** (p. 304) QoS policy on the corresponding factory for the **DDS::Entity** (p. 845).

By default, **ENTITY_FACTORY** (p. 304) is set so that it is not necessary to explicitly call **DDS::Entity::enable** (p. 848) on newly created entities.

The **DDS::Entity::enable** (p. 848) operation is idempotent. Calling enable on an already enabled **Entity** (p. 845) returns OK and has no effect.

If a **DDS::Entity** (p. 845) 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. 849)
- 'factory' operations
- **DDS::Entity::get_status_changes** (p. 850) 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_NotEnabled** (p. 1121).

It is legal to delete an **DDS::Entity** (p. 845) that has not been enabled by calling the proper operation on its factory.

Entities created from a factory that is disabled are created disabled, regardless of the setting of the **DDS::EntityFactoryQosPolicy** (p. 851).

Calling enable on an **Entity** (p. 845) whose factory is not enabled will fail and return **DDS::Retcode_PreconditionNotMet** (p. 1123).

If **DDS::EntityFactoryQosPolicy::autoenable_created_entities** (p. 852) is TRUE, the enable operation on a factory will automatically enable all entities created from that factory.

Listeners associated with an entity are not called until the entity is enabled.

Conditions associated with a disabled entity are "inactive," that is, they have a `trigger_value` == FALSE.

Exceptions:

One of the **Standard Return Codes** (p. 235), **Standard Return Codes** (p. 235) or **DDS::Retcode_PreconditionNotMet** (p. 1123).

Implements **DDS::Entity** (p. 848).

6.44.3.55 virtual StatusCondition ^ DDS::DomainParticipant::get_statuscondition () `[override, virtual]`

Allows access to the **DDS::StatusCondition** (p. 1183) associated with the **DDS::Entity** (p. 845).

The returned condition can then be added to a **DDS::WaitSet** (p. 1411) so that the application can wait for specific status changes that affect the **DDS::Entity** (p. 845).

Returns:

the status condition associated with this entity.

Implements **DDS::Entity** (p. 849).

6.44.3.56 virtual StatusMask DDS::DomainParticipant::get_status_changes () [override, virtual]

Retrieves the list of communication statuses in the **DDS::Entity** (p. 845) that are triggered.

That is, the list of statuses whose value has changed since the last time the application read the status using the get_*_status() method.

When the entity is first created or if the entity is not enabled, all communication statuses are in the "untriggered" state so the list returned by the get_status_changes operation will be empty.

The list of statuses returned by the get_status_changes operation refers to the status that are triggered on the **Entity** (p. 845) itself and does not include statuses that apply to contained entities.

Returns:

list of communication statuses in the **DDS::Entity** (p. 845) that are triggered.

See also:

Status Kinds (p. 238)

Implements **DDS::Entity** (p. 850).

6.44.3.57 virtual InstanceHandle_t DDS::DomainParticipant::get_instance_handle () [override, virtual]

Allows access to the **DDS::InstanceHandle_t** (p. 905) associated with the **DDS::Entity** (p. 845).

This operation returns the **DDS::InstanceHandle_t** (p. 905) that represents the **DDS::Entity** (p. 845).

Returns:

the instance handle associated with this entity.

Implements **DDS::Entity** (p. 850).

6.45 DDS::DomainParticipantFactory Class Reference

<<***singleton***>> *(p. 175)* <<***interface***>> *(p. 175)* Allows creation and destruction of **DDS::DomainParticipant** (p. 577) objects.

`#include <managed_domain.h>`

Public Member Functions

^ void **set_default_participant_qos** (**DomainParticipantQos**^ qos)

*Sets the default **DDS::DomainParticipantQos** (p. 683) values for this domain participant factory.*

^ void **set_default_participant_qos_with_profile** (System::String^ library_name, System::String^ profile_name)

<<**eXtension**>> (p. 174) *Sets the default **DDS::DomainParticipantQos** (p. 683) values for this domain participant factory based on the input XML QoS profile.*

^ void **get_default_participant_qos** (**DomainParticipantQos**^ qos)

*Initializes the **DDS::DomainParticipantQos** (p. 683) instance with default values.*

^ void **set_default_library** (System::String^ library_name)

<<**eXtension**>> (p. 174) *Sets the default XML library for a **DDS::DomainParticipantFactory** (p. 649).*

^ System::String^ **get_default_library** ()

<<**eXtension**>> (p. 174) *Gets the default XML library associated with a **DDS::DomainParticipantFactory** (p. 649).*

^ void **set_default_profile** (System::String^ library_name, System::String^ profile_name)

<<**eXtension**>> (p. 174) *Sets the default XML profile for a **DDS::DomainParticipantFactory** (p. 649).*

^ System::String^ **get_default_profile** ()

<<**eXtension**>> (p. 174) *Gets the default XML profile associated with a **DDS::DomainParticipantFactory** (p. 649).*

^ System::String^ **get_default_profile_library** ()

<<**eXtension**>> (p. 174) *Gets the library where the default XML profile is contained for a **DDS::DomainParticipantFactory** (p. 649).*

^ void **get_participant_qos_from_profile** (**DomainParticipantQos**^ qos, System::String^ library_name, System::String^ profile_name)

<<**eXtension**>> (p. 174) *Gets the* ***DDS::DomainParticipantQos*** *(p. 683) values associated with the input XML QoS profile.*

^ void **get_publisher_qos_from_profile** (**PublisherQos**^ qos, System::String^ library_name, System::String^ profile_name)

<<**eXtension**>> (p. 174) *Gets the* ***DDS::PublisherQos*** *(p. 1074) values associated with the input XML QoS profile.*

^ void **get_subscriber_qos_from_profile** (**SubscriberQos**^ qos, System::String^ library_name, System::String^ profile_name)

<<**eXtension**>> (p. 174) *Gets the* ***DDS::SubscriberQos*** *(p. 1230) values associated with the input XML QoS profile.*

^ void **get_datawriter_qos_from_profile** (**DataWriterQos**^ qos, System::String^ library_name, System::String^ profile_name)

<<**eXtension**>> (p. 174) *Gets the* ***DDS::DataWriterQos*** *(p. 546) values associated with the input XML QoS profile.*

^ void **get_datawriter_qos_from_profile_w_topic_name** (**DataWriterQos**^ qos, System::String^ library_name, System::String^ profile_name, System::String^ topic_name)

<<**eXtension**>> (p. 174) *Gets the* ***DDS::DataWriterQos*** *(p. 546) values associated with the input XML QoS profile while applying topic filters to the input topic name.*

^ void **get_datareader_qos_from_profile** (**DataReaderQos**^ qos, System::String^ library_name, System::String^ profile_name)

<<**eXtension**>> (p. 174) *Gets the* ***DDS::DataReaderQos*** *(p. 480) values associated with the input XML QoS profile.*

^ void **get_datareader_qos_from_profile_w_topic_name** (**DataReaderQos**^ qos, System::String^ library_name, System::String^ profile_name, System::String^ topic_name)

<<**eXtension**>> (p. 174) *Gets the* ***DDS::DataReaderQos*** *(p. 480) values associated with the input XML QoS profile while applying topic filters to the input topic name.*

^ void **get_topic_qos_from_profile** (**TopicQos**^ qos, System::String^ library_name, System::String^ profile_name)

<<**eXtension**>> (p. 174) *Gets the* ***DDS::TopicQos*** *(p. 1280) values associated with the input XML QoS profile.*

^ void **get_topic_qos_from_profile_w_topic_name** (**TopicQos**^ qos, System::String^ library_name, System::String^ profile_name, System::String^ topic_name)

<<**eXtension**>> (p. 174) *Gets the **DDS::TopicQos** (p. 1280) values associated with the input XML QoS profile while applying topic filters to the input topic name.*

^ void **get_qos_profile_libraries** (**StringSeq**^ library_names)

<<**eXtension**>> (p. 174) *Gets the names of all XML QoS profile libraries associated with the **DDS::DomainParticipantFactory** (p. 649)*

^ void **get_qos_profiles** (**StringSeq**^ profile_names, System::String^ library_name)

<<**eXtension**>> (p. 174) *Gets the names of all XML QoS profiles associated with the input XML QoS profile library.*

^ **DomainParticipant**^ **create_participant** (System::Int32 domainId, **DomainParticipantQos**^ qos, **DomainParticipantListener**^ listener, **StatusMask** mask)

*Creates a new **DDS::DomainParticipant** (p. 577) object.*

^ **DomainParticipant**^ **create_participant_with_profile** (System::Int32 domainId, System::String^ library_name, System::String^ profile_name, **DomainParticipantListener**^ listener, **StatusMask** mask)

<<**eXtension**>> (p. 174) *Creates a new **DDS::DomainParticipant** (p. 577) object using the **DDS::DomainParticipantQos** (p. 683) associated with the input XML QoS profile.*

^ void **delete_participant** (**DomainParticipant**^ %a_participant)

*Deletes an existing **DDS::DomainParticipant** (p. 577).*

^ **DomainParticipant**^ **lookup_participant** (Int32 domainId)

*Locates an existing **DDS::DomainParticipant** (p. 577).*

^ void **set_qos** (**DomainParticipantFactoryQos**^ qos)

Sets the value for a participant factory QoS.

^ void **get_qos** (**DomainParticipantFactoryQos**^ qos)

Gets the value for participant factory QoS.

^ void **load_profiles** ()

<<**eXtension**>> (p. 174) *Loads the XML QoS profiles.*

- void **reload_profiles** ()
  <<**eXtension**>> (p. 174) *Reloads the XML QoS profiles.*

- void **unload_profiles** ()
  <<**eXtension**>> (p. 174) *Unloads the XML QoS profiles.*

Static Public Member Functions

- static **DomainParticipantFactory**^ **get_instance** ()
  Gets the singleton instance of this class.

- static void **finalize_instance** ()
  <<**eXtension**>> (p. 174) *Destroys the singleton instance of this class.*

Properties

- static **DomainParticipantQos**^ **PARTICIPANT_QOS_DEFAULT** `[get]`
  Special value for creating a ***DomainParticipant*** *(p. 577) with default QoS.*

6.45.1 Detailed Description

<<***singleton***>> *(p. 175)* <<***interface***>> *(p. 175)* Allows creation and destruction of **DDS::DomainParticipant** (p. 577) objects.

The sole purpose of this class is to allow the creation and destruction of **DDS::DomainParticipant** (p. 577) objects. This class itself is a <<***singleton***>> *(p. 175)*, and accessed via the **get_instance()** (p. 653) method, and destroyed with **finalize_instance()** (p. 654) method.

A single application can participate in multiple domains by instantiating multiple **DDS::DomainParticipant** (p. 577) 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 index for each participant. (The participant index is a field in the **DDS::DiscoveryQosPolicy** (p. 571).) The port numbers used by a participant are calculated based on both the participant index and the domain ID, so if all participants on the same node have different participant indexes, they can coexist in the same domain.

- You cannot mix entities from different participants. For example, you cannot delete a topic on a different participant than you created it from, and you cannot ask a subscriber to create a reader for a topic created from a participant different than the subscriber's own participant. (Note that it is permissable for an application built on top of RTI Data Distribution Service 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. 577)

6.45.2 Member Function Documentation

6.45.2.1 static DomainParticipantFactory ^ DDS::DomainParticipantFactory::get_instance () [static]

Gets the singleton instance of this class.

Returns:

The **DDS::DomainParticipantFactory** (p. 649) instance.

MT Safety:

On non-Linux systems: UNSAFE for multiple threads to simultaneously make the FIRST call to either **DDS::DomainParticipantFactory::get_instance()** (p. 653) or **DDS::DomainParticipantFactory::finalize_instance()** (p. 654). Subsequent calls are thread safe. (On Linux systems, these calls are thread safe.)

DDS::TheParticipantFactory can be used as an alias for the singleton factory returned by this operation.

Returns:

The singleton **DDS::DomainParticipantFactory** (p. 649) instance.

See also:

DDS::TheParticipantFactory

Examples:

HelloWorld_publisher.cpp, and **HelloWorld_subscriber.cpp**.

6.45.2.2 static void DDS::DomainParticipantFactory::finalize_instance () [static]

<<eXtension>> *(p. 174)* Destroys the singleton instance of this class.

Only necessary to explicitly reclaim resources used by the participant factory singleton. Note that on many OSs, these resources are automatically reclaimed by the OS when the program terminates. However, some memory-check tools still flag these as unreclaimed. So this method provides a way to clean up memory used by the participant factory.

Precondition:

All participants created from the factory have been deleted.

Postcondition:

All resources belonging to the factory have been reclaimed. Another call to **DDS::DomainParticipantFactory::get_instance** (p. 653) will return a new lifecycle of the singleton.

MT Safety:

On non-Linux systems: UNSAFE for multiple threads to simultaneously make the FIRST call to either **DDS::DomainParticipantFactory::get_instance()** (p. 653) or **DDS::DomainParticipantFactory::finalize_instance()** (p. 654). Subsequent calls are thread safe. (On Linux systems, these calls are thread safe.)

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_PreconditionNotMet** (p. 1123)

6.45.2.3 void DDS::DomainParticipantFactory::set_default_participant_qos (DomainParticipantQos^ *qos*)

Sets the default **DDS::DomainParticipantQos** (p. 683) values for this domain participant factory.

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

MT Safety:

UNSAFE. It is not safe to retrieve the default QoS value from a domain participant factory while another thread may be simultaneously calling **DDS::DomainParticipantFactory::set_default_participant_qos** (p. 654)

Parameters:

qos <<*inout*>> *(p. 176)* Qos to be filled up. The special value **DDS::DomainParticipantFactory::PARTICIPANT_QOS_DEFAULT** (p. 35) may be passed as `qos` 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. 654) had never been called. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DomainParticipantFactory::PARTICIPANT_QOS_DEFAULT (p. 35)
DDS::DomainParticipantFactory::create_participant (p. 665)

6.45.2.4 void DDS::DomainParticipantFactory::set_default_participant_qos_with_profile (System::String^ *library_name*, System::String^ *profile_name*)

<<*eXtension*>> *(p. 174)* Sets the default **DDS::DomainParticipantQos** (p. 683) values for this domain participant factory based on the input XML QoS profile.

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

This default value will be used for newly created **DDS::DomainParticipant** (p. 577) if **DDS::DomainParticipantFactory::PARTICIPANT_QOS_DEFAULT** (p. 35) is specified as the `qos` parameter when **DDS::DomainParticipantFactory::create_participant** (p. 665) is called.

Precondition:

The **DDS::DomainParticipantQos** (p. 683) contained in the specified

XML QoS profile must be consistent, or else the operation will have no effect and fail with **DDS::Retcode_InconsistentPolicy** (p. 1119)

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. 654)

Parameters:

library_name *<<in>>* *(p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipantFactory::set_default_library** (p. 657)).

profile_name *<<in>>* *(p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipantFactory::set_default_profile** (p. 658)).

If the input profile cannot be found the method fails with **DDS::Retcode_Error** (p. 1116).

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_InconsistentPolicy** (p. 1119)

See also:

DDS::DomainParticipantFactory::PARTICIPANT_QOS_DEFAULT (p. 35)
DDS::DomainParticipantFactory::create_participant_with_profile (p. 667)

6.45.2.5 void DDS::DomainParticipantFactory::get_default_participant_qos (DomainParticipantQos^ *qos*)

Initializes the **DDS::DomainParticipantQos** (p. 683) 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. 654), or **DDS::DomainParticipantFactory::set_default_participant_qos_with_profile** (p. 655), or else, if the call was never made, the default values listed in **DDS::DomainParticipantQos** (p. 683).

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

Parameters:

qos *<<**out**>>* *(p. 176)* the domain participant's QoS Cannot be NULL.

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. 654)

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DomainParticipantFactory::PARTICIPANT_QOS_DEFAULT (p. 35)
DDS::DomainParticipantFactory::create_participant (p. 665)

6.45.2.6 void DDS::DomainParticipantFactory::set_default_library (System::String^ *library_name*)

*<<**eXtension**>>* *(p. 174)* Sets the default XML library for a **DDS::DomainParticipantFactory** (p. 649).

Any API requiring a library_name as a parameter can use null to refer to the default library.

See also:

DDS::DomainParticipantFactory::set_default_profile (p. 658) for more information.

Parameters:

library_name *<<**in**>>* *(p. 175)* Library name. If library_name is null any previous default is unset.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DomainParticipantFactory::get_default_library (p. 658)

6.45.2.7 System::String ^ DDS::DomainParticipantFactory::get_default_library ()

<<eXtension>> (p. 174) Gets the default XML library associated with a **DDS::DomainParticipantFactory** (p. 649).

Returns:

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

See also:

DDS::DomainParticipantFactory::set_default_library (p. 657)

6.45.2.8 void DDS::DomainParticipantFactory::set_default_profile (System::String^ *library_name*, System::String^ *profile_name*)

<<eXtension>> (p. 174) Sets the default XML profile for a **DDS::DomainParticipantFactory** (p. 649).

This method specifies the profile that will be used as the default the next time a default **DomainParticipantFactory** (p. 649) profile is needed during a call to a **DomainParticipantFactory** (p. 649) method. When calling a **DDS::DomainParticipantFactory** (p. 649) method that requires a `profile_name` parameter, you can use NULL to refer to the default profile. (This same information applies to setting a default library.)

This method does not set the default QoS for newly created DomainParticipants; for this functionality, use **DDS::DomainParticipantFactory::set_default_participant_qos_with_profile** (p. 655) (you may pass in NULL after having called **set_default_profile()** (p. 658)).

Parameters:

library_name *<<in>> (p. 175)* The library name containing the profile.

profile_name *<<in>> (p. 175)* The profile name. If profile_name is null any previous default is unset.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DomainParticipantFactory::get_default_profile (p. 659)
DDS::DomainParticipantFactory::get_default_profile_library (p. 659)

6.45.2.9 System::String ^ DDS::DomainParticipantFactory::get_default_profile ()

<<eXtension>> (p. 174) Gets the default XML profile associated with a **DDS::DomainParticipantFactory** (p. 649).

Returns:

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

See also:

DDS::DomainParticipantFactory::set_default_profile (p. 658)

6.45.2.10 System::String ^ DDS::DomainParticipantFactory::get_default_profile_library ()

<<eXtension>> (p. 174) Gets the library where the default XML profile is contained for a **DDS::DomainParticipantFactory** (p. 649).

The default profile library is automatically set when **DDS::DomainParticipantFactory::set_default_profile** (p. 658) is called.

This library can be different than the **DDS::DomainParticipantFactory** (p. 649) default library (see **DDS::DomainParticipantFactory::get_default_library** (p. 658)).

Returns:

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

See also:

DDS::DomainParticipantFactory::set_default_profile (p. 658)

6.45.2.11 void DDS::DomainParticipantFactory::get_participant_qos_from_profile (DomainParticipantQos^ *qos*, System::String^ *library_name*, System::String^ *profile_name*)

<<eXtension>> (p. 174) Gets the **DDS::DomainParticipantQos** (p. 683) values associated with the input XML QoS profile.

Parameters:

qos *<<out>> (p. 176)* Qos to be filled up. Cannot be NULL.

library_name *<<**in**>>* *(p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipantFactory::set_default_library** (p. 657)).

profile_name *<<**in**>>* *(p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipantFactory::set_default_profile** (p. 658)).

If the input profile cannot be found, the method fails with **DDS::Retcode_Error** (p. 1116).

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.45.2.12 void DDS::DomainParticipantFactory::get_publisher_qos_from_profile (PublisherQos^ *qos*, System::String^ *library_name*, System::String^ *profile_name*)

*<<**eXtension**>>* *(p. 174)* Gets the **DDS::PublisherQos** (p. 1074) values associated with the input XML QoS profile.

Parameters:

qos *<<**out**>>* *(p. 176)* Qos to be filled up. Cannot be NULL.

library_name *<<**in**>>* *(p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipantFactory::set_default_library** (p. 657)).

profile_name *<<**in**>>* *(p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipantFactory::set_default_profile** (p. 658)).

If the input profile cannot be found, the method fails with **DDS::Retcode_Error** (p. 1116).

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.45.2.13 void DDS::DomainParticipantFactory::get_subscriber_qos_from_profile (SubscriberQos^ *qos*, System::String^ *library_name*, System::String^ *profile_name*)

<<eXtension>> *(p. 174)* Gets the **DDS::SubscriberQos** (p. 1230) values associated with the input XML QoS profile.

Parameters:

qos *<<out>>* *(p. 176)* Qos to be filled up. Cannot be NULL.

library_name *<<in>>* *(p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipantFactory::set_default_library** (p. 657)).

profile_name *<<in>>* *(p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipantFactory::set_default_profile** (p. 658)).

If the input profile cannot be found, the method fails with **DDS::Retcode_Error** (p. 1116).

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.45.2.14 void DDS::DomainParticipantFactory::get_datawriter_qos_from_profile (DataWriterQos^ *qos*, System::String^ *library_name*, System::String^ *profile_name*)

<<eXtension>> *(p. 174)* Gets the **DDS::DataWriterQos** (p. 546) values associated with the input XML QoS profile.

Parameters:

qos *<<out>>* *(p. 176)* Qos to be filled up. Cannot be NULL.

library_name *<<in>>* *(p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipantFactory::set_default_library** (p. 657)).

profile_name *<<in>>* *(p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipantFactory::set_default_profile** (p. 658)).

If the input profile cannot be found, the method fails with **DDS::Retcode_Error** (p. 1116).

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.45.2.15 void DDS::DomainParticipantFactory::get_datawriter_qos_from_profile_w_topic_name (DataWriterQos^ *qos*, System::String^ *library_name*, System::String^ *profile_name*, System::String^ *topic_name*)

*<<**eXtension**>> (p. 174)* Gets the **DDS::DataWriterQos** (p. 546) values associated with the input XML QoS profile while applying topic filters to the input topic name.

Parameters:

qos *<<**out**>> (p. 176)* Qos to be filled up. Cannot be NULL.

library_name *<<**in**>> (p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipantFactory::set_default_library** (p. 657)).

profile_name *<<**in**>> (p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipantFactory::set_default_profile** (p. 658)).

topic_name *<<**in**>> (p. 175)* **Topic** (p. 1258) name that will be evaluated against the topic_filter attribute in the XML QoS profile. If topic_name is null, RTI Data Distribution Service will match only QoSs without explicit topic_filter expressions.

If the input profile cannot be found, the method fails with **DDS::Retcode_Error** (p. 1116).

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.45.2.16 void DDS::DomainParticipantFactory::get_datareader_qos_from_profile (DataReaderQos^ *qos*, System::String^ *library_name*, System::String^ *profile_name*)

*<<**eXtension**>> (p. 174)* Gets the **DDS::DataReaderQos** (p. 480) values associated with the input XML QoS profile.

Parameters:

qos *<<**out**>> (p. 176)* Qos to be filled up. Cannot be NULL.

library_name *<<in>> (p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipantFactory::set_default_library** (p. 657)).

profile_name *<<in>> (p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipantFactory::set_default_profile** (p. 658)).

If the input profile cannot be found, the method fails with **DDS::Retcode_Error** (p. 1116).

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.45.2.17 void DDS::DomainParticipantFactory::get_datareader_qos_from_profile_w_topic_name (DataReaderQos^ *qos*, System::String^ *library_name*, System::String^ *profile_name*, System::String^ *topic_name*)

<<eXtension>> (p. 174) Gets the **DDS::DataReaderQos** (p. 480) values associated with the input XML QoS profile while applying topic filters to the input topic name.

Parameters:

qos *<<out>> (p. 176)* Qos to be filled up. Cannot be NULL.

library_name *<<in>> (p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipantFactory::set_default_library** (p. 657)).

profile_name *<<in>> (p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipantFactory::set_default_profile** (p. 658)).

topic_name *<<in>> (p. 175)* **Topic** (p. 1258) name that will be evaluated against the topic_filter attribute in the XML QoS profile. If topic_name is null, RTI Data Distribution Service will match only QoSs without explicit topic_filter expressions.

If the input profile cannot be found, the method fails with **DDS::Retcode_Error** (p. 1116).

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.45.2.18 void DDS::DomainParticipantFactory::get_topic_qos_from_profile (TopicQos^ *qos*, System::String^ *library_name*, System::String^ *profile_name*)

<<eXtension>> (p. 174) Gets the **DDS::TopicQos** (p. 1280) values associated with the input XML QoS profile.

Parameters:

qos *<<out>> (p. 176)* Qos to be filled up. Cannot be NULL.

library_name *<<in>> (p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipantFactory::set_default_library** (p. 657)).

profile_name *<<in>> (p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipantFactory::set_default_profile** (p. 658)).

If the input profile cannot be found, the method fails with **DDS::Retcode_Error** (p. 1116).

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.45.2.19 void DDS::DomainParticipantFactory::get_topic_qos_from_profile_w_topic_name (TopicQos^ *qos*, System::String^ *library_name*, System::String^ *profile_name*, System::String^ *topic_name*)

<<eXtension>> (p. 174) Gets the **DDS::TopicQos** (p. 1280) values associated with the input XML QoS profile while applying topic filters to the input topic name.

Parameters:

qos *<<out>> (p. 176)* Qos to be filled up. Cannot be NULL.

library_name *<<in>> (p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipantFactory::set_default_library** (p. 657)).

profile_name *<<in>> (p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipantFactory::set_default_profile** (p. 658)).

topic_name *<<in>>* *(p. 175)* **Topic** (p. 1258) name that will be evaluated against the topic_filter attribute in the XML QoS profile. If topic_name is null, RTI Data Distribution Service will match only QoSs without explicit topic_filter expressions.

If the input profile cannot be found, the method fails with **DDS::Retcode_Error** (p. 1116).

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.45.2.20 void DDS::DomainParticipantFactory::get_qos_profile_libraries (StringSeq^ *library_names*)

<<eXtension>> *(p. 174)* Gets the names of all XML QoS profile libraries associated with the **DDS::DomainParticipantFactory** (p. 649)

Parameters:

library_names *<<out>>* *(p. 176)* **DDS::StringSeq** (p. 1192) to be filled with names of XML QoS profile libraries. Cannot be NULL.

6.45.2.21 void DDS::DomainParticipantFactory::get_qos_profiles (StringSeq^ *profile_names*, System::String^ *library_name*)

<<eXtension>> *(p. 174)* Gets the names of all XML QoS profiles associated with the input XML QoS profile library.

Parameters:

profile_names *<<out>>* *(p. 176)* **DDS::StringSeq** (p. 1192) to be filled with names of XML QoS profiles. Cannot be NULL.

library_name *<<in>>* *(p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipantFactory::set_default_library** (p. 657)).

6.45.2.22 DomainParticipant ^ DDS::DomainParticipantFactory::create_participant (System::Int32 *domainId*, DomainParticipantQos^ *qos*, DomainParticipantListener^ *listener*, StatusMask *mask*)

Creates a new **DDS::DomainParticipant** (p. 577) object.

Precondition:

The specified QoS policies must be consistent or the operation will fail and no **DDS::DomainParticipant** (p. 577) 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. 1423)). This in turn will ensure each participant uses a different port number (since the unicast port numbers are calculated from the participant index and the domain ID).

Note that if there is a single participant per host in a given domain, the participant index can be left at the default value (-1).

MT Safety:

UNSAFE. (1) If **DDS::DomainParticipantFactory::PARTICIPANT_QOS_DEFAULT** (p. 35) is used for `qos`, it is not safe to create the participant while another thread may simultaneously be calling **DDS::DomainParticipantFactory::set_default_participant_qos** (p. 654). (2) It is not safe to create one participant while another thread may simultaneously be looking up or deleting the same participant.

Parameters:

domainId *<<in>> (p. 175)* ID of the domain that the application intends to join. **[range]** [>=0], and does not violate guidelines stated in **DDS::RtpsWellKnownPorts_t** (p. 1142).

qos *<<in>> (p. 175)* the DomainParticipant's QoS. The special value **DDS::DomainParticipantFactory::PARTICIPANT_QOS_DEFAULT** (p. 35) can be used to indicate that the **DDS::DomainParticipant** (p. 577) should be created with the default **DDS::DomainParticipantQos** (p. 683) set in the **DDS::DomainParticipantFactory** (p. 649). Cannot be NULL.

listener *<<in>> (p. 175)* the domain participant's listener.

mask *<<in>> (p. 175)*. Changes of communication status to be invoked on the listener.

Returns:

domain participant or NULL on failure

See also:

Specifying QoS on entities (p. 267) for information on setting QoS before entity creation
DDS::DomainParticipantQos (p. 683) for rules on consistency among QoS

DDS::DomainParticipantFactory::PARTICIPANT_QOS_DEFAULT (p. 35)
NDDS_DISCOVERY_PEERS (p. 312)
DDS::DomainParticipantFactory::create_participant_with_profile() (p. 667)
DDS::DomainParticipantFactory::get_default_participant_qos() (p. 656)
DDS::DomainParticipant::set_listener() (p. 645)

6.45.2.23 DomainParticipant ^ DDS::DomainParticipantFactory::create_participant_with_profile (System::Int32 *domainId*, System::String^ *library_name*, System::String^ *profile_name*, DomainParticipantListener^ *listener*, StatusMask *mask*)

<<*eXtension*>> *(p. 174)* Creates a new **DDS::DomainParticipant** (p. 577) object using the **DDS::DomainParticipantQos** (p. 683) associated with the input XML QoS profile.

Precondition:

The **DDS::DomainParticipantQos** (p. 683) in the input profile must be consistent, or the operation will fail and no **DDS::DomainParticipant** (p. 577) 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. 1423)). This in turn will ensure each participant uses a different port number (since the unicast port numbers are calculated from the participant index and the domain ID).

Note that if there is a single participant per host in a given domain, the participant index can be left at the default value (-1).

MT Safety:

UNSAFE. (1) If **DDS::DomainParticipantFactory::PARTICIPANT_QOS_DEFAULT** (p. 35) is used for `qos`, it is not safe to create the participant while another thread may simultaneously be calling **DDS::DomainParticipantFactory::set_default_participant_qos** (p. 654). (2) It is not safe to create one participant while another thread may simultaneously be looking up or deleting the same participant.

Parameters:

domainId *<<in>>* (p. 175) ID of the domain that the application intends to join. **[range]** [>=0], and does not violate guidelines stated in **DDS::RtpsWellKnownPorts_t** (p. 1142).

library_name *<<in>>* (p. 175) Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipantFactory::set_default_library** (p. 657)).

profile_name *<<in>>* (p. 175) XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipantFactory::set_default_profile** (p. 658)).

listener *<<in>>* (p. 175) the DomainParticipant's listener.

mask *<<in>>* (p. 175). Changes of communication status to be invoked on the listener.

Returns:

domain participant or NULL on failure

See also:

Specifying QoS on entities (p. 267) for information on setting QoS before entity creation
DDS::DomainParticipantQos (p. 683) for rules on consistency among QoS
DDS::DomainParticipantFactory::PARTICIPANT_QOS_DEFAULT (p. 35)
NDDS_DISCOVERY_PEERS (p. 312)
DDS::DomainParticipantFactory::create_participant() (p. 665)
DDS::DomainParticipantFactory::get_default_participant_qos() (p. 656)
DDS::DomainParticipant::set_listener() (p. 645)

6.45.2.24 void DDS::DomainParticipantFactory::delete_participant (DomainParticipant^ % *a_participant*)

Deletes an existing **DDS::DomainParticipant** (p. 577).

Precondition:

All domain entities belonging to the participant must have already been deleted. Otherwise it fails with the error **DDS::Retcode_PreconditionNotMet** (p. 1123).

Postcondition:

Listener (p. 952) installed on the **DDS::DomainParticipant** (p. 577) will not be called after this method returns successfully.

Parameters:

a_participant *<<in>> (p. 175)* **DDS::DomainParticipant** (p. 577) to be deleted.

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_PreconditionNotMet** (p. 1123).

6.45.2.25 DomainParticipant ^ DDS::DomainParticipantFactory::lookup_participant (Int32 *domainId*)

Locates an existing **DDS::DomainParticipant** (p. 577).

If no such **DDS::DomainParticipant** (p. 577) exists, the operation will return NULL value.

If multiple **DDS::DomainParticipant** (p. 577) entities belonging to that domainId exist, then the operation will return one of them. It is not specified which one.

Parameters:

domainId *<<in>> (p. 175)* ID of the domain participant to lookup.

Returns:

domain participant if it exists, or NULL

6.45.2.26 void DDS::DomainParticipantFactory::set_qos (DomainParticipantFactoryQos^ *qos*)

Sets the value for a participant factory QoS.

The **DDS::DomainParticipantFactoryQos::entity_factory** (p. 673) can be changed. The other policies are immutable.

Note that despite having QoS, the **DDS::DomainParticipantFactory** (p. 649) is not an **DDS::Entity** (p. 845).

MT Safety:

UNSAFE. It is not safe to set the participant factory QoS while another thread may simultaneously be calling **DDS::DomainParticipantFactory::set_qos** (p. 669) or **DDS::DomainParticipantFactory::get_qos** (p. 670).

Parameters:

qos *<<**in**>>* *(p. 175)* Set of policies to be applied to **DDS::DomainParticipantFactory** (p. 649). Policies must be consistent. Immutable policies can only be changed before calling any other RTI Data Distribution Service methods except for **DDS::DomainParticipantFactory::get_qos** (p. 670) Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_ImmutablePolicy** (p. 1118) if immutable policy is changed, or **DDS::Retcode_InconsistentPolicy** (p. 1119) if policies are inconsistent

See also:

DDS::DomainParticipantFactoryQos (p. 673) for rules on consistency among QoS

6.45.2.27 void DDS::DomainParticipantFactory::get_qos (DomainParticipantFactoryQos^ *qos*)

Gets the value for participant factory QoS.

MT Safety:

UNSAFE. It is not safe to get the participant factory QoS while another thread may simultaneously be calling **DDS::DomainParticipantFactory::get_qos** (p. 670) or **DDS::DomainParticipantFactory::set_qos** (p. 669).

Parameters:

qos *<<**inout**>>* *(p. 176)* QoS to be filled up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.45.2.28 void DDS::DomainParticipantFactory::load_profiles ()

<<*eXtension*>> *(p. 174)* Loads the XML QoS profiles.

The XML QoS profiles are loaded implicitly after the first **DDS::DomainParticipant** (p. 577) is created or explicitly, after a call to this method.

This has the same effect as **DDS::DomainParticipantFactory::reload_profiles()** (p. 671).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::ProfileQosPolicy (p. 1019)

6.45.2.29 void DDS::DomainParticipantFactory::reload_profiles ()

<<*eXtension*>> *(p. 174)* Reloads the XML QoS profiles.

The XML QoS profiles are loaded implicitly after the first **DDS::DomainParticipant** (p. 577) is created or explicitly, after a call to this method.

This has the same effect as **DDS::DomainParticipantFactory::load_profiles()** (p. 671).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::ProfileQosPolicy (p. 1019)

6.45.2.30 void DDS::DomainParticipantFactory::unload_profiles ()

<<*eXtension*>> *(p. 174)* Unloads the XML QoS profiles.

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

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::ProfileQosPolicy (p. 1019)

6.46 DDS::DomainParticipantFactoryQos Class Reference

QoS policies supported by a **DDS::DomainParticipantFactory** (p. 649).

`#include <managed_domain.h>`

Public Attributes

- ^ **EntityFactoryQosPolicy entity_factory**
  Entity *(p. 845) factory policy,* ***ENTITY_FACTORY*** *(p. 304).*
- ^ **SystemResourceLimitsQosPolicy resource_limits**
  <<**eXtension**>> (p. 174) *System resource limits,* ***SYSTEM_RESOURCE_LIMITS*** *(p. 339).*
- ^ **ProfileQosPolicy**^ **profile**
  <<**eXtension**>> (p. 174) *Qos profile policy,* ***PROFILE*** *(p. 365).*

6.46.1 Detailed Description

QoS policies supported by a **DDS::DomainParticipantFactory** (p. 649).

Entity:

DDS::DomainParticipantFactory (p. 649)

See also:

QoS Policies (p. 260) and allowed ranges within each Qos.

6.46.2 Member Data Documentation

6.46.2.1 EntityFactoryQosPolicy DDS::DomainParticipantFactoryQos::entity_factory

Entity (p. 845) factory policy, **ENTITY_FACTORY** (p. 304).

6.46.2.2 SystemResourceLimitsQosPolicy DDS::DomainParticipantFactoryQos::resource_limits

<<***eXtension***>> *(p. 174)* System resource limits, **SYSTEM_RESOURCE_LIMITS** (p. 339).

6.46.2.3 ProfileQosPolicy ^ DDS::DomainParticipantFactoryQos::profile

<<*eXtension*>> *(*p. *174)* Qos profile policy, **PROFILE** (p. 365).

6.47 DDS::DomainParticipantListener Class Reference

<<***interface***>> *(p. 175)* **Listener** (p. 952) for participant status.

`#include <managed_domain.h>`

Inheritance diagram for DDS::DomainParticipantListener::

Public Member Functions

- ^ virtual void **on_inconsistent_topic** (**Topic**^ topic, **InconsistentTopicStatus**% status)

  Handle the DDS::StatusKind::INCONSISTENT_TOPIC_STATUS status.

- ^ virtual void **on_offered_deadline_missed** (**DataWriter**^ writer, **OfferedDeadlineMissedStatus**% status)

  Handles the DDS::StatusKind::OFFERED_DEADLINE_MISSED_STATUS status.

- ^ virtual void **on_liveliness_lost** (**DataWriter**^ writer, **LivelinessLostStatus**% status)

  Handles the DDS::StatusKind::LIVELINESS_LOST_STATUS status.

- ^ virtual void **on_offered_incompatible_qos** (**DataWriter**^ writer, **OfferedIncompatibleQosStatus**^ status)

  Handles the DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS status.

- ^ virtual void **on_publication_matched** (**DataWriter**^ writer, **PublicationMatchedStatus**% status)

  Handles the DDS::StatusKind::PUBLICATION_MATCHED_STATUS status.

- ^ virtual void **on_reliable_writer_cache_changed** (**DataWriter**^ writer, **ReliableWriterCacheChangedStatus**% status)

  <<**eXtension**>> (p. 174) *A change has occurred in the writer's cache of unacknowledged samples.*

- ^ virtual void **on_reliable_reader_activity_changed** (**DataWriter**^ writer, **ReliableReaderActivityChangedStatus**% status)

<<**eXtension**>> (p. 174) *A matched reliable reader has become active or become inactive.*

^ virtual void **on_requested_deadline_missed** (**DataReader**^ reader, **RequestedDeadlineMissedStatus**% status)

Handles the DDS::StatusKind::REQUESTED_DEADLINE_MISSED_STATUS communication status.

^ virtual void **on_liveliness_changed** (**DataReader**^ reader, **LivelinessChangedStatus**% status)

Handles the DDS::StatusKind::LIVELINESS_CHANGED_STATUS communication status.

^ virtual void **on_requested_incompatible_qos** (**DataReader**^ reader, **RequestedIncompatibleQosStatus**^ status)

Handles the DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS communication status.

^ virtual void **on_sample_rejected** (**DataReader**^ reader, **SampleRejectedStatus**% status)

Handles the DDS::StatusKind::SAMPLE_REJECTED_STATUS communication status.

^ virtual void **on_data_available** (**DataReader**^ reader)

Handle the DDS::StatusKind::DATA_AVAILABLE_STATUS communication status.

^ virtual void **on_sample_lost** (**DataReader**^ reader, **SampleLostStatus**% status)

Handles the DDS::StatusKind::SAMPLE_LOST_STATUS communication status.

^ virtual void **on_subscription_matched** (**DataReader**^ reader, **SubscriptionMatchedStatus**% status)

Handles the DDS::StatusKind::SUBSCRIPTION_MATCHED_STATUS communication status.

^ virtual void **on_data_on_readers** (**Subscriber**^ sub)

Handles the DDS::StatusKind::DATA_ON_READERS_STATUS communication status.

6.47.1 Detailed Description

<<***interface***>> *(p. 175)* **Listener** (p. 952) for participant status.

Entity:

DDS::DomainParticipant (p. 577)

Status:

Status Kinds (p. 238)

This is the interface that can be implemented by an application-provided class and then registered with the **DDS::DomainParticipant** (p. 577) such that the application can be notified by RTI Data Distribution Service of relevant status changes.

The **DDS::DomainParticipantListener** (p. 675) interface extends all other **Listener** (p. 952) interfaces and has no additional operation beyond the ones defined by the more general listeners.

The purpose of the **DDS::DomainParticipantListener** (p. 675) 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. 576) objects. When a relevant status change occurs, RTI Data Distribution Service will first attempt to notify the listener attached to the concerned **DDS::DomainEntity** (p. 576) if one is installed. Otherwise, RTI Data Distribution Service will notify the **Listener** (p. 952) attached to the **DDS::DomainParticipant** (p. 577).

Important: Because a **DDS::DomainParticipantListener** (p. 675) 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. 675) callback.

Note: Due to a thread-safety issue, the destruction of a **DomainParticipantListener** (p. 675) from an enabled **DomainParticipant** (p. 577) should be avoided even if the **DomainParticipantListener** (p. 675) has been removed from the **DomainParticipant** (p. 577). (This limitation does not affect the Java API.)

See also:

DDS::Listener (p. 952)
DDS::DomainParticipant::set_listener (p. 645)

6.47.2 Member Function Documentation

6.47.2.1 virtual void DDS::DomainParticipantListener::on_inconsistent_topic (Topic^ *topic*, InconsistentTopicStatus% *status*) [inline, virtual]

Handle the DDS::StatusKind::INCONSISTENT_TOPIC_STATUS status.

This callback is called when a remote **DDS::Topic** (p. 1258) is discovered but is inconsistent with the locally created **DDS::Topic** (p. 1258) of the same topic name.

Parameters:

topic <<*out*>> *(p. 176)* Locally created **DDS::Topic** (p. 1258) that triggers the listener callback

status <<*out*>> *(p. 176)* Current inconsistent status of locally created **DDS::Topic** (p. 1258)

Implements **DDS::TopicListener** (p. 1279).

6.47.2.2 virtual void DDS::DomainParticipantListener::on_offered_deadline_missed (DataWriter^ *writer*, OfferedDeadlineMissedStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::OFFERED_DEADLINE_MISSED_STATUS status.

This callback is called when the deadline that the **DDS::DataWriter** (p. 499) has committed through its **DEADLINE** (p. 281) qos policy was not respected for a specific instance. This callback is called for each deadline period elapsed during which the **DDS::DataWriter** (p. 499) failed to provide data for an instance.

Parameters:

writer <<*out*>> *(p. 176)* Locally created **DDS::DataWriter** (p. 499) that triggers the listener callback

status <<*out*>> *(p. 176)* Current deadline missed status of locally created **DDS::DataWriter** (p. 499)

Reimplemented from **DDS::PublisherListener** (p. 1070).

6.47.2.3 virtual void DDS::DomainParticipantListener::on_liveliness_lost (DataWriter^ *writer*, LivelinessLostStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::LIVELINESS_LOST_STATUS status.

This callback is called when the liveliness that the **DDS::DataWriter** (p. 499) has committed through its **LIVELINESS** (p. 286) qos policy was not respected; this **DDS::DataReader** (p. 433) entities will consider the **DDS::DataWriter** (p. 499) as no longer "alive/active". This callback will not be called when an already not alive **DDS::DataWriter** (p. 499) simply renames not alive for another liveliness period.

Parameters:

writer <<*out*>> *(p. 176)* Locally created **DDS::DataWriter** (p. 499) that triggers the listener callback

status <<*out*>> *(p. 176)* Current liveliness lost status of locally created **DDS::DataWriter** (p. 499)

Reimplemented from **DDS::PublisherListener** (p. 1071).

6.47.2.4 virtual void DDS::DomainParticipantListener::on_offered_incompatible_qos (DataWriter^ *writer*, OfferedIncompatibleQosStatus^ *status*) [inline, virtual]

Handles the DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS status.

This callback is called when the **DDS::DataWriterQos** (p. 546) of the **DDS::DataWriter** (p. 499) was incompatible with what was requested by a **DDS::DataReader** (p. 433). This callback is called when a **DDS::DataWriter** (p. 499) has discovered a **DDS::DataReader** (p. 433) for the same **DDS::Topic** (p. 1258) and common partition, but with a requested QoS that is incompatible with that offered by the **DDS::DataWriter** (p. 499).

Parameters:

writer <<*out*>> *(p. 176)* Locally created **DDS::DataWriter** (p. 499) that triggers the listener callback

status <<*out*>> *(p. 176)* Current incompatible qos status of locally created **DDS::DataWriter** (p. 499)

Reimplemented from **DDS::PublisherListener** (p. 1071).

6.47.2.5 virtual void DDS::DomainParticipantListener::on_publication_matched (DataWriter^ *writer*, PublicationMatchedStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::PUBLICATION_MATCHED_STATUS status.

This callback is called when the **DDS::DataWriter** (p. 499) has found a **DDS::DataReader** (p. 433) that matches the **DDS::Topic** (p. 1258), has a common partition and compatible QoS, or has ceased to be matched with a **DDS::DataReader** (p. 433) that was previously considered to be matched.

Parameters:

writer *<<**out**>>* *(p. 176)* Locally created **DDS::DataWriter** (p. 499) that triggers the listener callback

status *<<**out**>>* *(p. 176)* Current publication match status of locally created **DDS::DataWriter** (p. 499)

Reimplemented from **DDS::PublisherListener** (p. 1072).

6.47.2.6 virtual void DDS::DomainParticipantListener::on_reliable_writer_cache_changed (DataWriter^ *writer*, ReliableWriterCacheChangedStatus% *status*) [inline, virtual]

*<<**eXtension**>>* *(p. 174)* A change has occurred in the writer's cache of unacknowledged samples.

Parameters:

writer *<<**out**>>* *(p. 176)* Locally created **DDS::DataWriter** (p. 499) that triggers the listener callback

status *<<**out**>>* *(p. 176)* Current reliable writer cache changed status of locally created **DDS::DataWriter** (p. 499)

Reimplemented from **DDS::PublisherListener** (p. 1072).

6.47.2.7 virtual void DDS::DomainParticipantListener::on_reliable_reader_activity_changed (DataWriter^ *writer*, ReliableReaderActivityChangedStatus% *status*) [inline, virtual]

*<<**eXtension**>>* *(p. 174)* A matched reliable reader has become active or become inactive.

Parameters:

writer <<***out***>> *(*p. *176)* Locally created **DDS::DataWriter** (p. 499) that triggers the listener callback

status <<***out***>> *(*p. *176)* Current reliable reader activity changed status of locally created **DDS::DataWriter** (p. 499)

Reimplemented from **DDS::PublisherListener** (p. 1073).

6.47.2.8 virtual void DDS::DomainParticipantListener::on_requested_deadline_missed (DataReader^ *reader*, RequestedDeadlineMissedStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::REQUESTED_DEADLINE_MISSED_STATUS communication status.

Reimplemented from **DDS::SubscriberListener** (p. 1228).

6.47.2.9 virtual void DDS::DomainParticipantListener::on_liveliness_changed (DataReader^ *reader*, LivelinessChangedStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::LIVELINESS_CHANGED_STATUS communication status.

Reimplemented from **DDS::SubscriberListener** (p. 1228).

6.47.2.10 virtual void DDS::DomainParticipantListener::on_requested_incompatible_qos (DataReader^ *reader*, RequestedIncompatibleQosStatus^ *status*) [inline, virtual]

Handles the DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS communication status.

Reimplemented from **DDS::SubscriberListener** (p. 1228).

6.47.2.11 virtual void DDS::DomainParticipantListener::on_sample_rejected (DataReader^ *reader*, SampleRejectedStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::SAMPLE_REJECTED_STATUS communication status.

Reimplemented from **DDS::SubscriberListener** (p. 1228).

6.47.2.12 virtual void DDS::DomainParticipantListener::on_data_available (DataReader^ *reader*) [inline, virtual]

Handle the DDS::StatusKind::DATA_AVAILABLE_STATUS communication status.

Reimplemented from **DDS::SubscriberListener** (p. 1229).

6.47.2.13 virtual void DDS::DomainParticipantListener::on_sample_lost (DataReader^ *reader*, SampleLostStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::SAMPLE_LOST_STATUS communication status.

Reimplemented from **DDS::SubscriberListener** (p. 1229).

6.47.2.14 virtual void DDS::DomainParticipantListener::on_subscription_matched (DataReader^ *reader*, SubscriptionMatchedStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::SUBSCRIPTION_MATCHED_STATUS communication status.

Reimplemented from **DDS::SubscriberListener** (p. 1229).

6.47.2.15 virtual void DDS::DomainParticipantListener::on_data_on_readers (Subscriber^ *sub*) [inline, virtual]

Handles the DDS::StatusKind::DATA_ON_READERS_STATUS communication status.

Reimplemented from **DDS::SubscriberListener** (p. 1229).

6.48 DDS::DomainParticipantQos Class Reference

QoS policies supported by a **DDS::DomainParticipant** (p. 577) entity.

`#include <managed_domain.h>`

Public Attributes

- ^ **UserDataQosPolicy^ user_data**

  User data policy, ***USER_DATA*** *(p. 273).*

- ^ **EntityFactoryQosPolicy entity_factory**

  Entity *(p. 845) factory policy,* ***ENTITY_FACTORY*** *(p. 304).*

- ^ **WireProtocolQosPolicy wire_protocol**

  <<eXtension>> (p. 174) *Wire Protocol policy,* ***WIRE_PROTOCOL*** *(p. 325).*

- ^ **TransportBuiltinQosPolicy transport_builtin**

  <<eXtension>> (p. 174) *Transport Builtin policy,* ***TRANSPORT_BUILTIN*** *(p. 321).*

- ^ **TransportUnicastQosPolicy^ default_unicast**

  <<eXtension>> (p. 174) *Default Unicast Transport policy,* ***TRANSPORT_UNICAST*** *(p. 309).*

- ^ **DiscoveryQosPolicy^ discovery**

  <<eXtension>> (p. 174) *Discovery policy,* ***DISCOVERY*** *(p. 320).*

- ^ **DomainParticipantResourceLimitsQosPolicy resource_limits**

  <<eXtension>> (p. 174) *Domain participant resource limits policy,* ***DOMAIN_PARTICIPANT_RESOURCE_LIMITS*** *(p. 340).*

- ^ **EventQosPolicy^ event_qos**

  <<eXtension>> (p. 174) *Event policy,* ***EVENT*** *(p. 341).*

- ^ **ReceiverPoolQosPolicy^ receiver_pool**

  <<eXtension>> (p. 174) *Receiver pool policy,* ***RECEIVER_POOL*** *(p. 343).*

- ^ **DatabaseQosPolicy^ database**

  <<eXtension>> (p. 174) *Database policy,* ***DATABASE*** *(p. 342).*

^ **DiscoveryConfigQosPolicy^ discovery_config**

<<**eXtension**>> (p. 174) *Discovery config policy,* ***DISCOVERY_CONFIG*** *(p. 347).*

^ **PropertyQosPolicy^ property_qos**

<<**eXtension**>> (p. 174) *Property policy,* ***PROPERTY*** *(p. 357).*

^ **EntityNameQosPolicy^ participant_name**

<<**eXtension**>> (p. 174) *The participant name.* ***ENTITY_NAME*** *(p. 364)*

^ **TypeSupportQosPolicy type_support**

<<**eXtension**>> (p. 174) *Type support data,* ***TYPESUPPORT*** *(p. 350).*

6.48.1 Detailed Description

QoS policies supported by a **DDS::DomainParticipant** (p. 577) entity.

Certain members must be set in a consistent manner:

Length of **DDS::DomainParticipantQos::user_data** (p. 685) .value <= **DDS::DomainParticipantQos::resource_limits** (p. 686) .participant_user_data_max_length

For **DDS::DomainParticipantQos::discovery_config** (p. 686) .publication_writer

high_watermark <= **DDS::DomainParticipantQos::resource_limits** (p. 686) .local_writer_allocation .max_count heartbeats_per_max_samples <= **DDS::DomainParticipantQos::resource_limits** (p. 686) .local_writer_allocation.max_count

For **DDS::DomainParticipantQos::discovery_config** (p. 686) .suscription_writer

high_watermark <= **DDS::DomainParticipantQos::resource_limits** (p. 686) .local_reader_allocation.max_count heartbeats_per_max_samples <= **DDS::DomainParticipantQos::resource_limits** (p. 686) .local_reader_allocation.max_count

If any of the above are not true, **DDS::DomainParticipant::set_qos** (p. 642) and **DDS::DomainParticipant::set_qos_with_profile** (p. 643) and **DDS::DomainParticipantFactory::set_default_participant_qos** (p. 654) will fail with **DDS::Retcode_InconsistentPolicy** (p. 1119), and **DDS::DomainParticipantFactory::create_participant** (p. 665) will fail.

Entity:

DDS::DomainParticipant (p. 577)

See also:

QoS Policies (p. 260) and allowed ranges within each Qos.
NDDS_DISCOVERY_PEERS (p. 312)

6.48.2 Member Data Documentation

6.48.2.1 UserDataQosPolicy ^ DDS::DomainParticipantQos::user_data

User data policy, **USER_DATA** (p. 273).

6.48.2.2 EntityFactoryQosPolicy DDS::DomainParticipantQos::entity_factory

Entity (p. 845) factory policy, **ENTITY_FACTORY** (p. 304).

6.48.2.3 WireProtocolQosPolicy DDS::DomainParticipantQos::wire_protocol

<<*eXtension*>> *(p. 174)* Wire Protocol policy, **WIRE_PROTOCOL** (p. 325).

The wire protocol (RTPS) attributes associated with the participant.

6.48.2.4 TransportBuiltinQosPolicy DDS::DomainParticipantQos::transport_builtin

<<*eXtension*>> *(p. 174)* Transport Builtin policy, **TRANSPORT_BUILTIN** (p. 321).

6.48.2.5 TransportUnicastQosPolicy ^ DDS::DomainParticipantQos::default_unicast

<<*eXtension*>> *(p. 174)* Default Unicast Transport policy, **TRANSPORT_UNICAST** (p. 309).

6.48.2.6 DiscoveryQosPolicy ^ DDS::DomainParticipantQos::discovery

<<*eXtension*>> *(p. 174)* Discovery policy, **DISCOVERY** (p. 320).

6.48.2.7 DomainParticipantResourceLimitsQosPolicy DDS::DomainParticipantQos::resource_limits

<<*eXtension*>> *(p. 174)* Domain participant resource limits policy, **DOMAIN_PARTICIPANT_RESOURCE_LIMITS** (p. 340).

6.48.2.8 EventQosPolicy ^ DDS::DomainParticipantQos::event_qos

<<*eXtension*>> *(p. 174)* Event policy, **EVENT** (p. 341).

6.48.2.9 ReceiverPoolQosPolicy ^ DDS::DomainParticipantQos::receiver_pool

<<*eXtension*>> *(p. 174)* Receiver pool policy, **RECEIVER_POOL** (p. 343).

6.48.2.10 DatabaseQosPolicy ^ DDS::DomainParticipantQos::database

<<*eXtension*>> *(p. 174)* Database policy, **DATABASE** (p. 342).

6.48.2.11 DiscoveryConfigQosPolicy ^ DDS::DomainParticipantQos::discovery_config

<<*eXtension*>> *(p. 174)* Discovery config policy, **DISCOVERY_CONFIG** (p. 347).

6.48.2.12 PropertyQosPolicy ^ DDS::DomainParticipantQos::property_qos

<<*eXtension*>> *(p. 174)* Property policy, **PROPERTY** (p. 357).

6.48.2.13 EntityNameQosPolicy ^ DDS::DomainParticipantQos::participant_name

<<*eXtension*>> *(*p. *174**)* The participant name. **ENTITY_NAME** (p. 364)

6.48.2.14 TypeSupportQosPolicy DDS::DomainParticipantQos::type_support

<<*eXtension*>> *(*p. *174**)* Type support data, **TYPESUPPORT** (p. 350).

Optional value that is passed to a type plugin's on_participant_attached function.

6.49 DDS::DomainParticipantResourceLimitsQosPolicy Struct Reference

Various settings that configure how a **DDS::DomainParticipant** (p. 577) allocates and uses physical memory for internal resources, including the maximum sizes of various properties.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_domainparticipantresourcelimits_qos_policy_name** ()

  *Stringified human-readable name for **DDS::DomainParticipantResourceLimitsQosPolicy** (p. 688).*

Public Attributes

- **AllocationSettings_t local_writer_allocation**

  Allocation settings applied to local DataWriters.

- **AllocationSettings_t local_reader_allocation**

  Allocation settings applied to local DataReaders.

- **AllocationSettings_t local_publisher_allocation**

  *Allocation settings applied to local **Publisher** (p. 1044).*

- **AllocationSettings_t local_subscriber_allocation**

  *Allocation settings applied to local **Subscriber** (p. 1201).*

- **AllocationSettings_t local_topic_allocation**

  *Allocation settings applied to local **Topic** (p. 1258).*

- **AllocationSettings_t remote_writer_allocation**

  Allocation settings applied to remote DataWriters.

- **AllocationSettings_t remote_reader_allocation**

  Allocation settings applied to remote DataReaders.

- **AllocationSettings_t remote_participant_allocation**

  Allocation settings applied to remote DomainParticipants.

- **AllocationSettings_t matching_writer_reader_pair_allocation**

  Allocation settings applied to matching local writer and remote/local reader pairs.

- **AllocationSettings_t matching_reader_writer_pair_allocation**

  Allocation settings applied to matching local reader and remote/local writer pairs.

- **AllocationSettings_t ignored_entity_allocation**

  Allocation settings applied to ignored entities.

- **AllocationSettings_t content_filtered_topic_allocation**

  Allocation settings applied to content filtered topic.

- **AllocationSettings_t content_filter_allocation**

  Allocation settings applied to content filter.

- **AllocationSettings_t read_condition_allocation**

  Allocation settings applied to read condition pool.

- **AllocationSettings_t query_condition_allocation**

  Allocation settings applied to query condition pool.

- **AllocationSettings_t outstanding_asynchronous_sample_allocation**

  *Allocation settings applied to the maximum number of samples (from all **DDS::DataWriter** (p. 499)) waiting to be asynchronously written.*

- **AllocationSettings_t flow_controller_allocation**

  Allocation settings applied to flow controllers.

- System::Int32 **local_writer_hash_buckets**

  Hash_Buckets settings applied to local DataWriters.

- System::Int32 **local_reader_hash_buckets**

  Number of hash buckets for local DataReaders.

- System::Int32 **local_publisher_hash_buckets**

  *Number of hash buckets for local **Publisher** (p. 1044).*

- System::Int32 **local_subscriber_hash_buckets**

  *Number of hash buckets for local **Subscriber** (p. 1201).*

^ System::Int32 **local_topic_hash_buckets**

Number of hash buckets for local ***Topic*** *(*p. 1258*).*

^ System::Int32 **remote_writer_hash_buckets**

Number of hash buckets for remote DataWriters.

^ System::Int32 **remote_reader_hash_buckets**

Number of hash buckets for remote DataReaders.

^ System::Int32 **remote_participant_hash_buckets**

Number of hash buckets for remote DomainParticipants.

^ System::Int32 **matching_writer_reader_pair_hash_buckets**

Number of hash buckets for matching local writer and remote/local reader pairs.

^ System::Int32 **matching_reader_writer_pair_hash_buckets**

Number of hash buckets for matching local reader and remote/local writer pairs.

^ System::Int32 **ignored_entity_hash_buckets**

Number of hash buckets for ignored entities.

^ System::Int32 **content_filtered_topic_hash_buckets**

Number of hash buckets for content filtered topics.

^ System::Int32 **content_filter_hash_buckets**

Number of hash buckets for content filters.

^ System::Int32 **flow_controller_hash_buckets**

Number of hash buckets for flow controllers.

^ System::Int32 **max_gather_destinations**

Maximum number of destinations per RTI Data Distribution Service send.

^ System::Int32 **participant_user_data_max_length**

Maximum length of user data in ***DDS::DomainParticipantQos*** *(*p. 683*) and* ***DDS::ParticipantBuiltinTopicData*** *(*p. 1002*).*

^ System::Int32 **topic_data_max_length**

*Maximum length of topic data in **DDS::TopicQos*** *(p. 1280), **DDS::TopicBuiltinTopicData*** *(p. 1268), **DDS::PublicationBuiltinTopicData*** *(p. 1030) and **DDS::SubscriptionBuiltinTopicData*** *(p. 1233).*

^ System::Int32 **publisher_group_data_max_length**

*Maximum length of group data in **DDS::PublisherQos*** *(p. 1074) and **DDS::PublicationBuiltinTopicData*** *(p. 1030).*

^ System::Int32 **subscriber_group_data_max_length**

*Maximum length of group data in **DDS::SubscriberQos*** *(p. 1230) and **DDS::SubscriptionBuiltinTopicData*** *(p. 1233).*

^ System::Int32 **writer_user_data_max_length**

*Maximum length of user data in **DDS::DataWriterQos*** *(p. 546) and **DDS::PublicationBuiltinTopicData*** *(p. 1030).*

^ System::Int32 **reader_user_data_max_length**

*Maximum length of user data in **DDS::DataReaderQos*** *(p. 480) and **DDS::SubscriptionBuiltinTopicData*** *(p. 1233).*

^ System::Int32 **max_partitions**

*Maximum number of partition name strings allowable in a **DDS::PartitionQosPolicy*** *(p. 1008).*

^ System::Int32 **max_partition_cumulative_characters**

*Maximum number of combined characters allowable in all partition names in a **DDS::PartitionQosPolicy*** *(p. 1008).*

^ System::Int32 **type_code_max_serialized_length**

Maximum size of serialized string for type code.

^ System::Int32 **contentfilter_property_max_length**

This field is the maximum length of all data related to a Content-filtered topic.

^ System::Int32 **channel_seq_max_length**

*Maximum number of channels that can be specified in **DDS::MultiChannelQosPolicy*** *(p. 981) for MultiChannel DataWriters.*

^ System::Int32 **channel_filter_expression_max_length**

*Maximum length of a channel **DDS::ChannelSettings_t::filter_expression*** *(p. 403) in a MultiChannel **DataWriter*** *(p. 499).*

^ System::Int32 **participant_property_list_max_length**

*Maximum number of properties associated with the **DDS::DomainParticipant** (p. 577).*

^ System::Int32 **participant_property_string_max_length**

*Maximum string length of the properties associated with the **DDS::DomainParticipant** (p. 577).*

^ System::Int32 **writer_property_list_max_length**

*Maximum number of properties associated with a **DDS::DataWriter** (p. 499).*

^ System::Int32 **writer_property_string_max_length**

*Maximum string length of the properties associated with a **DDS::DataWriter** (p. 499).*

^ System::Int32 **reader_property_list_max_length**

*Maximum number of properties associated with a **DDS::DataReader** (p. 433).*

^ System::Int32 **reader_property_string_max_length**

*Maximum string length of the properties associated with a **DDS::DataReader** (p. 433).*

6.49.1 Detailed Description

Various settings that configure how a **DDS::DomainParticipant** (p. 577) 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. 577) is allowed to dynamically allocate memory as needed as users create local Entities such as **DDS::DataWriter** (p. 499) and **DDS::DataReader** (p. 433) objects or as the participant discovers new applications. By setting fixed values for the maximum parameters in this QoS policy, you can bound the memory that can be allocated by a **DDS::DomainParticipant** (p. 577). In addition, by setting the initial

values to the maximum values, you can prevent DomainParticipants from allocating memory after the initialization period.

The maximum sizes of different variable-length parameters such as the number of partitions that can be stored in the **DDS::PartitionQosPolicy** (p. 1008), the maximum length of data store in the **DDS::UserDataQosPolicy** (p. 1403) and **DDS::GroupDataQosPolicy** (p. 890), 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. 577) to another, a **DDS::DomainParticipant** (p. 577) 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_code_max_serialized_length** (p. 703).

This QoS policy is an extension to the DDS standard.

Entity:

DDS::DomainParticipant (p. 577)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)

6.49.2 Member Data Documentation

6.49.2.1 AllocationSettings_t DDS::DomainParticipantResourceLimitsQosPolicy::local_writer_allocation

Allocation settings applied to local DataWriters.

[**default**] initial_count = 16; max_count = DDS::LENGTH_UNLIMITED; incremental_count = -1

[**range**] See allowed ranges in struct **DDS::AllocationSettings_t** (p. 369)

6.49.2.2 AllocationSettings_t DDS::DomainParticipantResourceLimitsQosPolicy::local_reader_allocation

Allocation settings applied to local DataReaders.

[**default**] initial_count = 16; max_count = DDS::LENGTH_UNLIMITED; incremental_count = -1

[**range**] See allowed ranges in struct **DDS::AllocationSettings_t** (p. 369)

6.49.2.3 AllocationSettings_t DDS::DomainParticipantResourceLimitsQosPolicy::local_publisher_allocation

Allocation settings applied to local **Publisher** (p. 1044).

[**default**] initial_count = 4; max_count = DDS::LENGTH_UNLIMITED; incremental_count = -1

[**range**] See allowed ranges in struct **DDS::AllocationSettings_t** (p. 369)

6.49.2.4 AllocationSettings_t DDS::DomainParticipantResourceLimitsQosPolicy::local_subscriber_allocation

Allocation settings applied to local **Subscriber** (p. 1201).

[**default**] initial_count = 4; max_count = DDS::LENGTH_UNLIMITED; incremental_count = -1

[**range**] See allowed ranges in struct **DDS::AllocationSettings_t** (p. 369)

6.49.2.5 AllocationSettings_t DDS::DomainParticipantResourceLimitsQosPolicy::local_topic_allocation

Allocation settings applied to local **Topic** (p. 1258).

[**default**] initial_count = 16; max_count = DDS::LENGTH_UNLIMITED; incremental_count = -1

[**range**] See allowed ranges in struct **DDS::AllocationSettings_t** (p. 369)

6.49.2.6 AllocationSettings_t DDS::DomainParticipantResourceLimitsQosPolicy::remote_writer_allocation

Allocation settings applied to remote DataWriters.

Remote DataWriters include all DataWriters, both local and remote.

[**default**] initial_count = 64; max_count = DDS::LENGTH_UNLIMITED; incremental_count = -1

[**range**] See allowed ranges in struct **DDS::AllocationSettings_t** (p. 369)

6.49.2.7 AllocationSettings_t DDS::DomainParticipantResourceLimitsQosPolicy::remote_reader_allocation

Allocation settings applied to remote DataReaders.

Remote DataReaders include all DataReaders, both local and remote.

[**default**] initial_count = 64; max_count = DDS::LENGTH_UNLIMITED; incremental_count = -1

[**range**] See allowed ranges in struct **DDS::AllocationSettings_t** (p. 369)

6.49.2.8 AllocationSettings_t DDS::DomainParticipantResourceLimitsQosPolicy::remote_participant_allocation

Allocation settings applied to remote DomainParticipants.

Remote DomainParticipants include all DomainParticipants, both local and remote.

[**default**] initial_count = 16; max_count = DDS::LENGTH_UNLIMITED; incremental_count = -1

[**range**] See allowed ranges in struct **DDS::AllocationSettings_t** (p. 369)

6.49.2.9 AllocationSettings_t DDS::DomainParticipantResourceLimitsQosPolicy::matching_writer_reader_pair_allocation

Allocation settings applied to matching local writer and remote/local reader pairs.

[**default**] initial_count = 32; max_count = DDS::LENGTH_UNLIMITED; incremental_count = -1

[**range**] See allowed ranges in struct **DDS::AllocationSettings_t** (p. 369)

6.49.2.10 AllocationSettings_t DDS::DomainParticipantResourceLimitsQosPolicy::matching_reader_writer_pair_allocation

Allocation settings applied to matching local reader and remote/local writer pairs.

[**default**] initial_count = 32; max_count = DDS::LENGTH_UNLIMITED; incremental_count = -1

[**range**] See allowed ranges in struct **DDS::AllocationSettings_t** (p. 369)

6.49.2.11 AllocationSettings_t DDS::DomainParticipantResourceLimitsQosPolicy::ignored_entity_allocation

Allocation settings applied to ignored entities.

[**default**] initial_count = 8; max_count = DDS::LENGTH_UNLIMITED; incremental_count = -1

[**range**] See allowed ranges in struct **DDS::AllocationSettings_t** (p. 369)

6.49.2.12 AllocationSettings_t DDS::DomainParticipantResourceLimitsQosPolicy::content_filtered_topic_allocation

Allocation settings applied to content filtered topic.

[**default**] initial_count = 4; max_count = DDS::LENGTH_UNLIMITED; incremental_count = -1

[**range**] See allowed ranges in struct **DDS::AllocationSettings_t** (p. 369)

6.49.2.13 AllocationSettings_t DDS::DomainParticipantResourceLimitsQosPolicy::content_filter_allocation

Allocation settings applied to content filter.

[**default**] initial_count = 4; max_count = DDS::LENGTH_UNLIMITED; incremental_count = -1

[**range**] See allowed ranges in struct **DDS::AllocationSettings_t** (p. 369)

6.49.2.14 AllocationSettings_t DDS::DomainParticipantResourceLimitsQosPolicy::read_condition_allocation

Allocation settings applied to read condition pool.

[**default**] initial_count = 4; max_count = DDS::LENGTH_UNLIMITED, incremental_count = -1

[**range**] See allowed ranges in struct **DDS::AllocationSettings_t** (p. 369)

6.49.2.15 AllocationSettings_t DDS::DomainParticipantResourceLimitsQosPolicy::query_condition_allocation

Allocation settings applied to query condition pool.

[**default**] initial_count = 4; max_count = DDS::LENGTH_UNLIMITED, incremental_count = -1

[**range**] See allowed ranges in struct **DDS::AllocationSettings_t** (p. 369)

6.49.2.16 AllocationSettings_t DDS::DomainParticipantResourceLimitsQosPolicy::outstanding_asynchronous_sample_allocation

Allocation settings applied to the maximum number of samples (from all **DDS::DataWriter** (p. 499)) waiting to be asynchronously written.

[**default**] initial_count = 64; max_count = DDS::LENGTH_UNLIMITED, incremental_count = -1

[**range**] See allowed ranges in struct **DDS::AllocationSettings_t** (p. 369)

6.49.2.17 AllocationSettings_t DDS::DomainParticipantResourceLimitsQosPolicy::flow_controller_allocation

Allocation settings applied to flow controllers.

[**default**] initial_count = 4; max_count = DDS::LENGTH_UNLIMITED, incremental_count = -1

[**range**] See allowed ranges in struct **DDS::AllocationSettings_t** (p. 369)

6.49.2.18 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::local_writer_hash_buckets

Hash_Buckets settings applied to local DataWriters.

[**default**] 4

[**range**] [1, 10000]

6.49.2.19 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::local_reader_hash_buckets

Number of hash buckets for local DataReaders.

[default] 4

[range] [1, 10000]

6.49.2.20 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::local_publisher_hash_buckets

Number of hash buckets for local **Publisher** (p. 1044).

[default] 1

[range] [1, 10000]

6.49.2.21 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::local_subscriber_hash_buckets

Number of hash buckets for local **Subscriber** (p. 1201).

[default] 1

[range] [1, 10000]

6.49.2.22 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::local_topic_hash_buckets

Number of hash buckets for local **Topic** (p. 1258).

[default] 4

[range] [1, 10000]

6.49.2.23 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::remote_writer_hash_buckets

Number of hash buckets for remote DataWriters.

Remote DataWriters include all DataWriters, both local and remote.

[**default**] 16

[**range**] [1, 10000]

6.49.2.24 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::remote_reader_hash_buckets

Number of hash buckets for remote DataReaders.

Remote DataReaders include all DataReaders, both local and remote.

[**default**] 16

[**range**] [1, 10000]

6.49.2.25 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::remote_participant_hash_buckets

Number of hash buckets for remote DomainParticipants.

Remote DomainParticipants include all DomainParticipants, both local and remote.

[**default**] 4

[**range**] [1, 10000]

6.49.2.26 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::matching_writer_reader_pair_hash_buckets

Number of hash buckets for matching local writer and remote/local reader pairs.

[**default**] 32

[**range**] [1, 10000]

6.49.2.27 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::matching_reader_writer_pair_hash_buckets

Number of hash buckets for matching local reader and remote/local writer pairs.

[**default**] 32

[**range**] [1, 10000]

6.49.2.28 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::ignored_entity_hash_buckets

Number of hash buckets for ignored entities.

[default] 1

[range] [1, 10000]

6.49.2.29 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::content_filtered_topic_hash_buckets

Number of hash buckets for content filtered topics.

[default] 1

[range] [1, 10000]

6.49.2.30 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::content_filter_hash_buckets

Number of hash buckets for content filters.

[default] 1

[range] [1, 10000]

6.49.2.31 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::flow_controller_hash_buckets

Number of hash buckets for flow controllers.

[default] 1

[range] [1, 10000]

6.49.2.32 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::max_gather_destinations

Maximum number of destinations per RTI Data Distribution Service send.

When RTI Data Distribution Service sends out a message, it has the capability to send to multiple destinations to be more efficient. The maximum number

of destinations per RTI Data Distribution Service send is specified by max_gather_destinations.

[**default**] 8

[**range**] [4, 1 million]

6.49.2.33 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::participant_user_data_max_length

Maximum length of user data in **DDS::DomainParticipantQos** (p. 683) and **DDS::ParticipantBuiltinTopicData** (p. 1002).

[**default**] 256

[**range**] [0,0x7fffffff]

6.49.2.34 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::topic_data_max_length

Maximum length of topic data in **DDS::TopicQos** (p. 1280), **DDS::TopicBuiltinTopicData** (p. 1268), **DDS::PublicationBuiltinTopicData** (p. 1030) and **DDS::SubscriptionBuiltinTopicData** (p. 1233).

[**default**] 256

[**range**] [0,0x7fffffff]

6.49.2.35 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::publisher_group_data_max_length

Maximum length of group data in **DDS::PublisherQos** (p. 1074) and **DDS::PublicationBuiltinTopicData** (p. 1030).

[**default**] 256

[**range**] [0,0x7fffffff]

6.49.2.36 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::subscriber_group_data_max_length

Maximum length of group data in **DDS::SubscriberQos** (p. 1230) and **DDS::SubscriptionBuiltinTopicData** (p. 1233).

[default] 256

[range] [0,0x7fffffff]

6.49.2.37 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::writer_user_data_max_length

Maximum length of user data in **DDS::DataWriterQos** (p. 546) and **DDS::PublicationBuiltinTopicData** (p. 1030).

[default] 256

[range] [0,0x7fffffff]

6.49.2.38 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::reader_user_data_max_length

Maximum length of user data in **DDS::DataReaderQos** (p. 480) and **DDS::SubscriptionBuiltinTopicData** (p. 1233).

[default] 256

[range] [0,0x7fffffff]

6.49.2.39 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::max_partitions

Maximum number of partition name strings allowable in a **DDS::PartitionQosPolicy** (p. 1008).

This value cannot exceed 64.

[default] 64

[range] [0,64]

6.49.2.40 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::max_partition_cumulative_characters

Maximum number of combined characters allowable in all partition names in a **DDS::PartitionQosPolicy** (p. 1008).

The maximum number of combined characters should account for a terminating NULL ('\0') character for each partition name string.

This value cannot exceed 256.

[**default**] 256

[**range**] [0,256]

6.49.2.41 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::type_code_max_serialized_length

Maximum size of serialized string for type code.

This parameter limits the size of the type code that a **DDS::DomainParticipant** (p. 577) 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 type codes larger than the default maximum of 2048 bytes. So it is common for users to set this parameter to a larger value. 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.

[**default**] 2048

[**range**] [0,0xffff]

6.49.2.42 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::contentfilter_property_max_length

This field is the maximum length of all data related to a Content-filtered topic.

This is the sum of the length of the content filter 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]

6.49.2.43 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::channel_seq_max_length

Maximum number of channels that can be specified in **DDS::MultiChannelQosPolicy** (p. 981) for MultiChannel DataWriters.

[default] 32

[range] [0,0xffff]

6.49.2.44 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::channel_filter_expression_max_length

Maximum length of a channel **DDS::ChannelSettings_t::filter_expression** (p. 403) in a MultiChannel **DataWriter** (p. 499).

The length should account for a terminating NULL ('\0') character.

[default] 256

[range] [0,0xffff]

6.49.2.45 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::participant_property_list_max_length

Maximum number of properties associated with the **DDS::DomainParticipant** (p. 577).

[default] 32

[range] [0,0xffff]

6.49.2.46 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::participant_property_string_max_length

Maximum string length of the properties associated with the **DDS::DomainParticipant** (p. 577).

The string length is defined as the cumulative length in bytes of all the pair

(name,value) associated with the **DDS::DomainParticipant** (p. 577) properties.

[default] 1024

[range] [0,0xffff]

6.49.2.47 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::writer_property_list_max_length

Maximum number of properties associated with a **DDS::DataWriter** (p. 499).

[range] [0,0xffff]

[default] 32

6.49.2.48 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::writer_property_string_max_length

Maximum string length of the properties associated with a **DDS::DataWriter** (p. 499).

The string length is defined as the cumulative length in bytes of all the pair (name,value) associated with the data writer properties.

[default] 1024

[range] [0,0xffff]

6.49.2.49 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::reader_property_list_max_length

Maximum number of properties associated with a **DDS::DataReader** (p. 433).

[default] 32

[range] [0,0xffff]

6.49.2.50 System::Int32 DDS::DomainParticipantResourceLimitsQosPolicy::reader_property_string_max_length

Maximum string length of the properties associated with a **DDS::DataReader** (p. 433).

The string length is defined as the cumulative length in bytes of all the pair (name,value) associated with a **DDS::DataReader** (p. 433) properties.

[default] 1024

[range] [0,0xffff]

6.50 DDS::DoubleSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < System::Double >.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::DoubleSeq::

Public Member Functions

- ^ **DoubleSeq** ()
  Constructs an empty sequence of doubles with an initial maximum of zero.
- ^ **DoubleSeq** (System::Int32 max)
  Constructs an empty sequence of doubles with the given initial maximum.
- ^ **DoubleSeq** (**DoubleSeq**^ doubles)
  Constructs a new sequence containing the given doubles.

6.50.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < System::Double >.

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

System::Double
DDS::Sequence (p. 1163)

6.50.2 Constructor & Destructor Documentation

6.50.2.1 DDS::DoubleSeq::DoubleSeq () `[inline]`

Constructs an empty sequence of doubles with an initial maximum of zero.

6.50.2.2 DDS::DoubleSeq::DoubleSeq (System::Int32 *max*) `[inline]`

Constructs an empty sequence of doubles with the given initial maximum.

6.50.2.3 DDS::DoubleSeq::DoubleSeq (DoubleSeq^ *doubles*) [inline]

Constructs a new sequence containing the given doubles.

Parameters:

doubles the initial contents of this sequence

6.51 DDS::DurabilityQosPolicy Struct Reference

This QoS policy specifies whether or not RTI Data Distribution Service will store and deliver previously published data samples to new **DDS::DataReader** (p. 433) entities that join the network later.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_durability_qos_policy_name** ()

  *Stringified human-readable name for **DDS::DurabilityQosPolicy** (p. 709).*

Public Attributes

- **DurabilityQosPolicyKind kind**

  The kind of durability.

Properties

- System::Boolean **direct_communication** `[get, set]`

  <<**eXtension**>> (p. 174) *Indicates whether or not a TRANSIENT or PERSISTENT **DDS::DataReader** (p. 433) should receive samples directly from a TRANSIENT or PERSISTENT **DDS::DataWriter** (p. 499)*

6.51.1 Detailed Description

This QoS policy specifies whether or not RTI Data Distribution Service will store and deliver previously published data samples to new **DDS::DataReader** (p. 433) entities that join the network later.

Entity:

DDS::Topic (p. 1258), **DDS::DataReader** (p. 433), **DDS::DataWriter** (p. 499)

Status:

DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS,
DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS

Properties:

RxO (p. 268) = YES
Changeable (p. 269) = **UNTIL ENABLE** (p. 269)

See also:

DURABILITY_SERVICE (p. 297)

6.51.2 Usage

It is possible for a **DDS::DataWriter** (p. 499) to start publishing data before all (or any) **DDS::DataReader** (p. 433) entities have joined the network.

Moreover, a **DDS::DataReader** (p. 433) 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. 433) 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 Data Distribution Service will maintain internally. That is, RTI Data Distribution Service 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. 276) policy is set to DDS::DurabilityQosPolicyKind::VOLATILE_DURABILITY_QOS.

6.51.2.1 Transient and Persistent Durability

For the purpose of implementing the DURABILITY QoS kind TRANSIENT or PERSISTENT, RTI Data Distribution Service behaves *as if* for each **Topic** (p. 1258) that has **DDS::DurabilityQosPolicy::kind** (p. 712) of DDS::DurabilityQosPolicyKind::TRANSIENT_DURABILITY_QOS or DDS::DurabilityQosPolicyKind::PERSISTENT_DURABILITY_QOS there is a corresponding "built-in" **DDS::DataReader** (p. 433) and **DDS::DataWriter** (p. 499) configured with the same DURABILITY kind. In other words, it is *as if* somewhere in the system, independent of the original **DDS::DataWriter** (p. 499), there is a built-in durable **DDS::DataReader** (p. 433) subscribing to that **Topic** (p. 1258) and a built-in durable **DataWriter** (p. 499) 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 appli-

cation's **DDS::DataWriter** (p. 499) and **DDS::DataReader** (p. 433) entities. For each transient or persistent **DDS::Topic** (p. 1258), the built-in fictitious Persistence Service **DDS::DataReader** (p. 433) and **DDS::DataWriter** (p. 499) have their QoS configured from the QoS of your application's **DDS::DataWriter** (p. 499) and **DDS::DataReader** (p. 433) entities that communicate on that **DDS::Topic** (p. 1258).

For a given **DDS::Topic** (p. 1258), the usual request/offered semantics apply to the matching between any **DDS::DataWriter** (p. 499) in the domain that writes the **DDS::Topic** (p. 1258) and the built-in transient/persistent **DDS::DataReader** (p. 433) for that **DDS::Topic** (p. 1258); similarly for the built-in transient/persistent **DDS::DataWriter** (p. 499) for a **DDS::Topic** (p. 1258) and any **DDS::DataReader** (p. 433) for the **DDS::Topic** (p. 1258). As a consequence, a **DDS::DataWriter** (p. 499) that has an incompatible QoS will not send its data to the *RTI Persistence Service*, and a **DDS::DataReader** (p. 433) that has an incompatible QoS will not get data from it.

Incompatibilities between local **DDS::DataReader** (p. 433) and **DDS::DataWriter** (p. 499) entities and the corresponding fictitious built-in transient/persistent entities cause the DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS and DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS to change and the corresponding **Listener** (p. 952) invocations and/or signaling of **DDS::Condition** (p. 408) objects as they would with your application's own entities.

The value of **DDS::DurabilityServiceQosPolicy::service_cleanup_delay** (p. 716) 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

2. While in the NOT_ALIVE_DISPOSED state, the system detects that there are no more 'live' **DDS::DataWriter** (p. 499) entities writing the instance. That is, all existing writers either unregister the instance (call unregister) or lose their liveliness,

and

3. A time interval longer that **DDS::DurabilityServiceQosPolicy::service_cleanup_delay** (p. 716) has elapsed since the moment RTI Data Distribution Service detected that the previous two conditions were met.

The utility of **DDS::DurabilityServiceQosPolicy::service_cleanup_delay** (p. 716) 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.

6.51.3 Compatibility

The value offered is considered compatible with the value requested if and only if the inequality *offered kind >= requested kind* evaluates to 'TRUE'. For the purposes of this inequality, the values of DURABILITY kind are considered ordered such that DDS::DurabilityQosPolicyKind::VOLATILE_DURABILITY_QOS < DDS::DurabilityQosPolicyKind::TRANSIENT_LOCAL_DURABILITY_QOS < DDS::DurabilityQosPolicyKind::TRANSIENT_DURABILITY_QOS < DDS::DurabilityQosPolicyKind::PERSISTENT_DURABILITY_QOS.

6.51.4 Member Data Documentation

6.51.4.1 DurabilityQosPolicyKind DDS::DurabilityQosPolicy::kind

The kind of durability.

[default] DDS::DurabilityQosPolicyKind::VOLATILE_DURABILITY_QOS

6.51.5 Property Documentation

6.51.5.1 System:: Boolean DDS::DurabilityQosPolicy::direct_communication [get, set]

<<***eXtension***>> *(p. 174)* Indicates whether or not a TRANSIENT or PERSISTENT **DDS::DataReader** (p. 433) should receive samples directly from a TRANSIENT or PERSISTENT **DDS::DataWriter** (p. 499)

When direct_communication is set to true, a TRANSIENT or PERSISTENT **DDS::DataReader** (p. 433) will receive samples from both the original **DDS::DataWriter** (p. 499) configured with TRANSIENT or PERSISTENT durability and the **DDS::DataWriter** (p. 499) 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. 499) and the persistence service, the middleware will discard the duplicate.

When direct_communication is set to false, a TRANSIENT or PERSISTENT **DDS::DataReader** (p. 433) will only receive samples from the

DDS::DataWriter (p. 499) created by the persistence service. This brokered communication pattern provides a way to guarantee eventual consistency.

[**default**] true

6.52 DDS::DurabilityServiceQosPolicy Struct Reference

Various settings to configure the external *RTI Persistence Service* used by RTI Data Distribution Service for DataWriters with a **DDS::DurabilityQosPolicy** (p. 709) setting of DDS::DurabilityQosPolicyKind::PERSISTENT_DURABILITY_QOS or DDS::DurabilityQosPolicyKind::TRANSIENT_DURABILITY_QOS.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_durabilityservice_qos_policy_name** ()

  *Stringified human-readable name for **DDS::DurabilityServiceQosPolicy** (p. 714).*

Public Attributes

- **Duration_t service_cleanup_delay**

  *[**Not supported (optional)**] Controls when the service is able to remove all information regarding a data instances.*

- **HistoryQosPolicyKind history_kind**

  The kind of history to apply in recouping durable data.

- System::Int32 **history_depth**

  Part of history QoS policy to apply when feeding a late joiner.

- System::Int32 **max_samples**

  Part of resource limits QoS policy to apply when feeding a late joiner.

- System::Int32 **max_instances**

  Part of resource limits QoS policy to apply when feeding a late joiner.

- System::Int32 **max_samples_per_instance**

  Part of resource limits QoS policy to apply when feeding a late joiner.

6.52.1 Detailed Description

Various settings to configure the external *RTI Persistence Service* used by RTI Data Distribution Service for DataWriters with a **DDS::DurabilityQosPolicy** (p. 709) setting of DDS::DurabilityQosPolicyKind::PERSISTENT_DURABILITY_QOS or DDS::DurabilityQosPolicyKind::TRANSIENT_DURABILITY_QOS.

Entity:

DDS::Topic (p. 1258), **DDS::DataWriter** (p. 499)

Properties:

RxO (p. 268) = NO
Changeable (p. 269) = **UNTIL ENABLE** (p. 269)

See also:

DURABILITY (p. 276)
HISTORY (p. 294)
RESOURCE_LIMITS (p. 298)

6.52.2 Usage

When a DataWriter's **DDS::DurabilityQosPolicy::kind** (p. 712) is DDS::DurabilityQosPolicyKind::PERSISTENT_DURABILITY_QOS or DDS::DurabilityQosPolicyKind::TRANSIENT_DURABILITY_QOS, an external service, the *RTI Persistence Service*, is used to store and possibly forward the data sent by the **DDS::DataWriter** (p. 499) to **DDS::DataReader** (p. 433) 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. 499), such as how much data to store. For example, it configures the **HISTORY** (p. 294) and the **RESOURCE_LIMITS** (p. 298) used by the fictitious **DataReader** (p. 433) and **DataWriter** (p. 499) 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.

6.52.3 Member Data Documentation

6.52.3.1 Duration_t DDS::DurabilityServiceQosPolicy::service_cleanup_delay

[**Not supported (optional)**] Controls when the service is able to remove all information regarding a data instances.

[**default**] 0

6.52.3.2 HistoryQosPolicyKind DDS::DurabilityServiceQosPolicy::history_kind

The kind of history to apply in recouping durable data.

[**default**] DDS::HistoryQosPolicyKind::KEEP_LAST_HISTORY_QOS

6.52.3.3 System::Int32 DDS::DurabilityServiceQosPolicy::history_depth

Part of history QoS policy to apply when feeding a late joiner.

[**default**] 1

6.52.3.4 System::Int32 DDS::DurabilityServiceQosPolicy::max_samples

Part of resource limits QoS policy to apply when feeding a late joiner.

[**default**] DDS::LENGTH_UNLIMITED

6.52.3.5 System::Int32 DDS::DurabilityServiceQosPolicy::max_instances

Part of resource limits QoS policy to apply when feeding a late joiner.

[**default**] DDS::LENGTH_UNLIMITED

6.52.3.6 System::Int32 DDS::DurabilityServiceQosPolicy::max_samples_per_instance

Part of resource limits QoS policy to apply when feeding a late joiner.

[**default**] DDS::LENGTH_UNLIMITED

6.53 DDS::Duration_t Struct Reference

Type for *duration* representation.

`#include <managed_infrastructure.h>`

Public Member Functions

- System::Boolean **is_infinite** ()
- System::Boolean **is_zero** ()

Public Attributes

- System::Int32 **sec**

  seconds

- System::UInt32 **nanosec**

  nanoseconds

Properties

- static System::Int32 **DURATION_ZERO_SEC** `[get]`

  A zero-length second period of time.

- static System::Int32 **DURATION_ZERO_NSEC** `[get]`

  A zero-length nano-second period of time.

- static System::Int32 **DURATION_INFINITE_SEC** `[get]`

  An infinite second period of time.

- static System::Int32 **DURATION_INFINITE_NSEC** `[get]`

  An infinite nano-second period of time.

- static **Duration_t DURATION_INFINITE** `[get]`

  An infinite period of time.

- static **Duration_t DURATION_ZERO** `[get]`

  A zero-length period of time.

6.53.1 Detailed Description

Type for *duration* representation.

Represents a time interval.

6.53.2 Member Data Documentation

6.53.2.1 System::Int32 DDS::Duration_t::sec

seconds

6.53.2.2 System::UInt32 DDS::Duration_t::nanosec

nanoseconds

6.54 DDS::DynamicData Class Reference

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

`#include <managed_dynamicdata.h>`

Inheritance diagram for DDS::DynamicData::

Public Member Functions

- virtual System::Boolean **copy_from** (**DynamicData**^ other)

  Deeply copy from the given object to this object.

- **DynamicData** (**DDS::TypeCode**^ type, **DynamicDataProperty_t**^ property)

  *The constructor for new **DDS::DynamicData** (p. 719) objects.*

- System::Boolean **is_valid** ()

  Indicates whether the object was constructed properly.

- void **copy** (**DynamicData**^ src)

  Deeply copy from the given object to this object.

- System::Boolean **equal** (**DynamicData**^ other)

  *Indicate whether the contents of another **DDS::DynamicData** (p. 719) sample are the same as those of this one.*

- void **clear_all_members** ()

  Clear the contents of all data members of this object, including key members.

- void **clear_nonkey_members** ()

  Clear the contents of all data members of this object, not including key members.

- void **clear_member** (System::String^ member_name, System::Int32 member_id)

  Clear the contents of a single data member of this object.

- void **get_info** (**DynamicDataInfo**^ info_out)

  *Fill in the given descriptor with information about this **DDS::DynamicData** (p. 719).*

^ void **bind_type** (**DDS::TypeCode**^ type)

*If this **DDS::DynamicData** (p. 719) object is not yet associated with a data type, set that type now to the given **DDS::TypeCode** (p. 1301).*

^ void **unbind_type** ()

*Dissociate this **DDS::DynamicData** (p. 719) object from any particular data type.*

^ void **bind_complex_member** (**DynamicData**^ value_out, System::String^ member_name, System::Int32 member_id)

*Use another **DDS::DynamicData** (p. 719) object to provide access to a complex field of this **DDS::DynamicData** (p. 719) object.*

^ void **unbind_complex_member** (**DynamicData**^ value)

*Tear down the association created by a **DDS::DynamicData::bind_complex_member** (p. 740) operation, committing any changes to the outer object since then.*

^ **DDS::TypeCode**^ **get_type** ()

*Get the data type, of which this **DDS::DynamicData** (p. 719) represents an instance.*

^ **TCKind get_type_kind** ()

Get the kind of this object's data type.

^ System::UInt32 **get_member_count** ()

Get the number of members in this sample.

^ System::Boolean **member_exists** (System::String^ member_name, System::Int32 member_id)

Indicates whether a member of a particular name/ID exists in this data sample.

^ System::Boolean **member_exists_in_type** (System::String^ member_name, System::Int32 member_id)

Indicates whether a member of a particular name/ID exists in this data sample's type.

^ void **get_member_info** (**DynamicDataMemberInfo**^ info, System::String^ member_name, System::Int32 member_id)

*Fill in the given descriptor with information about the identified member of this **DDS::DynamicData** (p. 719) sample.*

^ void **get_member_info_by_index** (**DynamicDataMemberInfo**^ info, System::UInt32 index)

*Fill in the given descriptor with information about the identified member of this **DDS::DynamicData** (p. 719) sample.*

^ void **get_member_type** (**DDS::TypeCode**^ %type_out, System::String^ member_name, System::Int32 member_id)

Get the type of the given member of this sample.

^ System::Boolean **is_member_key** (System::String^ member_name, System::Int32 member_id)

Indicates whether a given member forms part of the key of this sample's data type.

^ System::Int32 **get_int** (System::String^ member_name, System::Int32 member_id)

Get the value of the given field, which is of type System::Int32 or another type implicitly convertible to it.

^ System::Int16 **get_short** (System::String^ member_name, System::Int32 member_id)

Get the value of the given field, which is of type System::Int16 or another type implicitly convertible to it.

^ System::UInt32 **get_uint** (System::String^ member_name, System::Int32 member_id)

Get the value of the given field, which is of type System::UInt32 or another type implicitly convertible to it.

^ System::UInt16 **get_ushort** (System::String^ member_name, System::Int32 member_id)

Get the value of the given field, which is of type System::UInt16 or another type implicitly convertible to it.

^ System::Single **get_float** (System::String^ member_name, System::Int32 member_id)

Get the value of the given field, which is of type System::Single or another type implicitly convertible to it.

^ System::Double **get_double** (System::String^ member_name, System::Int32 member_id)

Get the value of the given field, which is of type System::Double or another type implicitly convertible to it.

^ System::Boolean **get_boolean** (System::String^ member_name, System::Int32 member_id)

Get the value of the given field, which is of type System::Boolean.

^ System::Char **get_char** (System::String^ member_name, System::Int32 member_id)

Get the value of the given field, which is of type System::Char or another type implicitly convertible to it.

^ System::Byte **get_byte** (System::String^ member_name, System::Int32 member_id)

Get the value of the given field, which is of type System::Byte or another type implicitly convertible to it.

^ System::Int64 **get_long** (System::String^ member_name, System::Int32 member_id)

Get the value of the given field, which is of type System::Int64 or another type implicitly convertible to it.

^ System::UInt64 **get_ulong** (System::String^ member_name, System::Int32 member_id)

Get the value of the given field, which is of type System::UInt64 or another type implicitly convertible to it.

^ **LongDouble get_longdouble** (System::String^ member_name, System::Int32 member_id)

*Get the value of the given field, which is of type **DDS::LongDouble** (p. 976) or another type implicitly convertible to it.*

^ System::Char **get_wchar** (System::String^ member_name, System::Int32 member_id)

Get the value of the given field, which is of type System::Char or another type implicitly convertible to it.

^ System::String^ **get_string** (System::String^ member_name, System::Int32 member_id)

Get the value of the given field, which is of type System::String.

^ System::String^ **get_wstring** (System::String^ member_name, System::Int32 member_id)

Get the value of the given field, which is of type System::String.

^ void **get_complex_member** (**DynamicData**^ value_out, System::String^ member_name, System::Int32 member_id)

Get a copy of the value of the given field, which is of some composed type.

^ void **get_int_array** (array< System::Int32 >^array, System::String^ member_name, System::Int32 member_id)

Get a copy of the given array member.

^ void **get_short_array** (array< System::Int16 >^array, System::String^ member_name, System::Int32 member_id)

Get a copy of the given array member.

^ void **get_uint_array** (array< System::UInt32 >^array, System::String^ member_name, System::Int32 member_id)

Get a copy of the given array member.

^ void **get_ushort_array** (array< System::UInt16 >^array, System::String^ member_name, System::Int32 member_id)

Get a copy of the given array member.

^ void **get_float_array** (array< System::Single >^array, System::String^ member_name, System::Int32 member_id)

Get a copy of the given array member.

^ void **get_double_array** (array< System::Double >^array, System::String^ member_name, System::Int32 member_id)

Get a copy of the given array member.

^ void **get_boolean_array** (array< System::Boolean >^array, System::String^ member_name, System::Int32 member_id)

Get a copy of the given array member.

^ void **get_char_array** (array< System::Char >^array, System::String^ member_name, System::Int32 member_id)

Get a copy of the given array member.

^ void **get_byte_array** (array< System::Byte >^array, System::String^ member_name, System::Int32 member_id)

Get a copy of the given array member.

^ void **get_long_array** (array< System::Int64 >^array, System::String^ member_name, System::Int32 member_id)

Get a copy of the given array member.

^ void **get_ulong_array** (array< System::UInt64 >^array, System::String^ member_name, System::Int32 member_id)

Get a copy of the given array member.

^ void **get_longdouble_array** (array< **LongDouble** >^array, System::String^ member_name, System::Int32 member_id)

Get a copy of the given array member.

^ void **get_wchar_array** (array< System::Char >^array, System::String^ member_name, System::Int32 member_id)

Get a copy of the given array member.

^ void **get_int_seq** (**IntSeq**^ seq, System::String^ member_name, System::Int32 member_id)

Get a copy of the given sequence member.

^ void **get_short_seq** (**ShortSeq**^ seq, System::String^ member_name, System::Int32 member_id)

Get a copy of the given sequence member.

^ void **get_uint_seq** (**UnsignedIntSeq**^ seq, System::String^ member_name, System::Int32 member_id)

Get a copy of the given sequence member.

^ void **get_ushort_seq** (**UnsignedShortSeq**^ seq, System::String^ member_name, System::Int32 member_id)

Get a copy of the given sequence member.

^ void **get_float_seq** (**FloatSeq**^ seq, System::String^ member_name, System::Int32 member_id)

Get a copy of the given sequence member.

^ void **get_double_seq** (**DoubleSeq**^ seq, System::String^ member_name, System::Int32 member_id)

Get a copy of the given sequence member.

^ void **get_boolean_seq** (**BooleanSeq**^ seq, System::String^ member_name, System::Int32 member_id)

Get a copy of the given sequence member.

^ void **get_char_seq** (**CharSeq**^ seq, System::String^ member_name, System::Int32 member_id)

Get a copy of the given sequence member.

^ void **get_byte_seq** (**ByteSeq**^ seq, System::String^ member_name, System::Int32 member_id)

Get a copy of the given sequence member.

^ void **get_long_seq** (**LongSeq**^ seq, System::String^ member_name, System::Int32 member_id)

Get a copy of the given sequence member.

^ void **get_ulong_seq** (**UnsignedLongSeq**^ seq, System::String^ member_name, System::Int32 member_id)

Get a copy of the given sequence member.

^ void **get_longdouble_seq** (**LongDoubleSeq**^ seq, System::String^ member_name, System::Int32 member_id)

Get a copy of the given sequence member.

^ void **get_wchar_seq** (**WcharSeq**^ seq, System::String^ member_name, System::Int32 member_id)

Get a copy of the given sequence member.

^ void **set_int** (System::String^ member_name, System::Int32 member_id, System::Int32 value)

Set the value of the given field, which is of type System::Int32.

^ void **set_short** (System::String^ member_name, System::Int32 member_id, System::Int16 value)

Set the value of the given field, which is of type System::Int16.

^ void **set_uint** (System::String^ member_name, System::Int32 member_id, System::UInt32 value)

Set the value of the given field, which is of type System::UInt32.

^ void **set_ushort** (System::String^ member_name, System::Int32 member_id, System::UInt16 value)

Set the value of the given field, which is of type System::UInt16.

^ void **set_float** (System::String^ member_name, System::Int32 member_id, System::Single value)

Set the value of the given field, which is of type System::Single.

^ void **set_double** (System::String^ member_name, System::Int32 member_id, System::Double value)

Set the value of the given field, which is of type System::Double.

^ void **set_boolean** (System::String^ member_name, System::Int32 member_id, System::Boolean value)

Set the value of the given field, which is of type System::Boolean.

^ void **set_char** (System::String^ member_name, System::Int32 member_id, System::Char value)

Set the value of the given field, which is of type System::Char.

^ void **set_byte** (System::String^ member_name, System::Int32 member_id, System::Byte value)

Set the value of the given field, which is of type System::Byte.

^ void **set_long** (System::String^ member_name, System::Int32 member_id, System::Int64 value)

Set the value of the given field, which is of type System::Int64.

^ void **set_ulong** (System::String^ member_name, System::Int32 member_id, System::UInt64 value)

Set the value of the given field, which is of type System::UInt64.

^ void **set_longdouble** (System::String^ member_name, System::Int32 member_id, **LongDouble** value)

*Set the value of the given field, which is of type **DDS::LongDouble** (*p. 976*).*

^ void **set_wchar** (System::String^ member_name, System::Int32 member_id, System::Char value)

Set the value of the given field, which is of type System::Char.

^ void **set_string** (System::String^ member_name, System::Int32 member_id, System::String^ value)

Set the value of the given field of type System::String.

^ void **set_wstring** (System::String^ member_name, System::Int32 member_id, System::String^ value)

Set the value of the given field of type System::String.

^ void **set_complex_member** (System::String^ member_name, System::Int32 member_id, **DynamicData**^ value)

*Copy the state of the given **DDS::DynamicData** (*p. 719*) object into a member of this object.*

^ void **set_int_array** (System::String^ member_name, System::Int32 member_id, array< System::Int32 >^array)

Set the contents of the given array member.

^ void **set_short_array** (System::String^ member_name, System::Int32 member_id, array< System::Int16 >^array)

Set the contents of the given array member.

^ void **set_uint_array** (System::String^ member_name, System::Int32 member_id, array< System::UInt32 >^array)

Set the contents of the given array member.

^ void **set_ushort_array** (System::String^ member_name, System::Int32 member_id, array< System::UInt16 >^array)

Set the contents of the given array member.

^ void **set_float_array** (System::String^ member_name, System::Int32 member_id, array< System::Single >^array)

Set the contents of the given array member.

^ void **set_double_array** (System::String^ member_name, System::Int32 member_id, array< System::Double >^array)

Set the contents of the given array member.

^ void **set_boolean_array** (System::String^ member_name, System::Int32 member_id, array< System::Boolean >^array)

Set the contents of the given array member.

^ void **set_char_array** (System::String^ member_name, System::Int32 member_id, array< System::Char >^array)

Set the contents of the given array member.

^ void **set_byte_array** (System::String^ member_name, System::Int32 member_id, array< System::Byte >^array)

Set the contents of the given array member.

^ void **set_long_array** (System::String^ member_name, System::Int32 member_id, array< System::Int64 >^array)

Set the contents of the given array member.

^ void **set_ulong_array** (System::String^ member_name, System::Int32 member_id, array< System::UInt64 >^array)

Set the contents of the given array member.

^ void **set_longdouble_array** (System::String^ member_name, System::Int32 member_id, array< **LongDouble** >^array)

Set the contents of the given array member.

^ void **set_wchar_array** (System::String^ member_name, System::Int32 member_id, array< System::Char >^array)

Set the contents of the given array member.

^ void **set_int_seq** (System::String^ member_name, System::Int32 member_id, **IntSeq**^ value)

Set the contents of the given sequence member.

^ void **set_short_seq** (System::String^ member_name, System::Int32 member_id, **ShortSeq**^ value)

Set the contents of the given sequence member.

^ void **set_uint_seq** (System::String^ member_name, System::Int32 member_id, **UnsignedIntSeq**^ value)

Set the contents of the given sequence member.

^ void **set_ushort_seq** (System::String^ member_name, System::Int32 member_id, **UnsignedShortSeq**^ value)

Set the contents of the given sequence member.

^ void **set_float_seq** (System::String^ member_name, System::Int32 member_id, **FloatSeq**^ value)

Set the contents of the given sequence member.

^ void **set_double_seq** (System::String^ member_name, System::Int32 member_id, **DoubleSeq**^ value)

Set the contents of the given sequence member.

^ void **set_boolean_seq** (System::String^ member_name, System::Int32 member_id, **BooleanSeq**^ value)

Set the contents of the given sequence member.

^ void **set_char_seq** (System::String^ member_name, System::Int32 member_id, **CharSeq**^ value)

Set the contents of the given sequence member.

^ void **set_byte_seq** (System::String^ member_name, System::Int32 member_id, **ByteSeq**^ value)

Set the contents of the given sequence member.

^ void **set_long_seq** (System::String^ member_name, System::Int32 member_id, **LongSeq**^ value)

Set the contents of the given sequence member.

^ void **set_ulong_seq** (System::String^ member_name, System::Int32 member_id, **UnsignedLongSeq**^ value)

Set the contents of the given sequence member.

^ void **set_longdouble_seq** (System::String^ member_name, System::Int32 member_id, **LongDoubleSeq**^ value)

Set the contents of the given sequence member.

^ void **set_wchar_seq** (System::String^ member_name, System::Int32 member_id, **WcharSeq**^ value)

Set the contents of the given sequence member.

^ **~DynamicData** ()

*Finalize and deallocate this **DDS::DynamicData** (p. 719) sample.*

Static Public Attributes

^ static System::Int32 **MEMBER_ID_UNSPECIFIED**

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

^ static **DynamicDataProperty_t**^ **DYNAMIC_DATA_PROPERTY_DEFAULT**

*Sentinel constant indicating default values for **DDS::DynamicDataProperty_t** (p. 813).*

6.54.1 Detailed Description

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

Objects of type **DDS::DynamicData** (p. 719) represent corresponding objects of the type identified by their **DDS::TypeCode** (p. 1301). 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. 719) objects can represent any complex data type, including those of type kinds DDS::TCKind::TK_ARRAY, DDS::TCKind::TK_SEQUENCE, DDS::TCKind::TK_STRUCT, DDS::TCKind::TK_UNION, DDS::TCKind::TK_VALUE, and DDS::TCKind::TK_SPARSE. 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.

6.54.2 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. When you define a sparse type, you will typically choose both explicitly. 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 MyType {
     short theShort;
     long theLong;
 };
```

IDs work the same way for **DDS::DynamicData** (p. 719) objects representing arrays and sequences, since the elements of these collections have no explicit IDs: the ID is one more than the index. (The first element is ID 1, the second is 2, etc.) Array and sequence elements do not have names.

Multi-dimensional arrays are effectively flattened by the **DDS::DynamicData** (p. 719) API. 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.

For unions (DDS::TCKind::TK_UNION), the ID of a member is the discriminator value corresponding to that member.

6.54.3 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.

6.54.3.1 Lifecycle and Utility Methods

Managing the lifecycle of **DDS::DynamicData** (p. 719) objects is simple. You have two choices:

1. Usually, you will go through a **DDS::DynamicDataTypeSupport** (p. 822) factory object, which will ensure that the type and property information for the new **DDS::DynamicData** (p. 719) 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. 719) 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.

DDS::DynamicDataTypeSupport (p. 822)	**DDS::DynamicData** (p. 719)
DDS::DynamicDataTypeSupport::create_data (p. 826)	**DDS::DynamicData::DynamicData** (p. 734)

Table 6.1: Lifecycle

You can also copy **DDS::DynamicData** (p. 719) objects:

- **DDS::DynamicData::copy** (p. 736)

You can test them for equality:

- **DDS::DynamicData::equal** (p. 736)

6.54.3.2 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.

When working with a **DDS::DynamicData** (p. 719) 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_NoData** (p. 1120). Calling "set" for an index that is past the end of a sequence will cause that sequence to automatically lengthen (filling with default contents).

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_member** (p. 737)
- **DDS::DynamicData::clear_all_members** (p. 737)
- **DDS::DynamicData::clear_nonkey_members** (p. 737)

6.54.3.3 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. 743)
- **DDS::DynamicData::get_type_kind** (p. 743)
- **DDS::DynamicData::get_member_type** (p. 746)
- **DDS::DynamicData::get_member_info** (p. 745)
- **DDS::DynamicData::get_member_count** (p. 743)
- **DDS::DynamicData::get_member_info_by_index** (p. 746)
- **DDS::DynamicData::member_exists** (p. 744)
- **DDS::DynamicData::member_exists_in_type** (p. 744)
- **DDS::DynamicData::is_member_key** (p. 747)

6.54.3.4 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. 719) 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. 719) object as a view into an enclosing **DDS::DynamicData** (p. 719) object.

- **DDS::DynamicData::bind_type** (p. 738)
- **DDS::DynamicData::unbind_type** (p. 739)
- **DDS::DynamicData::bind_complex_member** (p. 740)
- **DDS::DynamicData::unbind_complex_member** (p. 742)

6.54.3.5 Keys

Keys can be specified in dynamically defined types just as they can in types defined in generated code. However, there are some minor restrictions when *sparse value types* are involved (see DDS::TCKind::TK_SPARSE).

- If a type has a member that is of a sparse value type, that member cannot be a key for the enclosing type.
- Sparse value types themselves may have at most a single key field. That field may itself be of any type.

6.54.4 Performance

Due to the way in which **DDS::DynamicData** (p. 719) objects manage their internal state, it is typically more efficient, when setting the field values of a **DDS::DynamicData** (p. 719) for the first time, to do so in the declared order of those fields.

For example, suppose a type definition like the following:

```
struct MyType {
     float my_float;
     sequence<octet> my_bytes;
     short my_short;
 };
```

The richness of the type system makes it difficult to fully characterize the performance differences between all access patterns. Nevertheless, the following are generally true:

- It will be most performant to set the value of `my_float`, then `my_bytes`, and finally `my_short`.
- The order of modification has a greater impact for types of kind DDS::TCKind::TK_STRUCT and DDS::TCKind::TK_VALUE than it does for types of kind DDS::TCKind::TK_SPARSE.
- Modifications to variable-sized types (i.e. those containing strings, sequences, unions, or optional members) are more expensive than modifications to fixed-size types.

MT Safety:

UNSAFE. In general, using a single **DDS::DynamicData** (p. 719) object concurrently from multiple threads is *unsafe*.

6.54.5 Constructor & Destructor Documentation

6.54.5.1 DDS::DynamicData::DynamicData (DDS::TypeCode^ *type*, DynamicDataProperty_t^ *property*)

The constructor for new **DDS::DynamicData** (p. 719) objects.

The type parameter may be null. In that case, this **DDS::DynamicData** (p. 719) must be *bound* with **DDS::DynamicData::bind_type** (p. 738) or **DDS::DynamicData::bind_complex_member** (p. 740) before it can be used.

If the **DDS::TypeCode** (p. 1301) is not null, the newly constructed **DDS::DynamicData** (p. 719) object will retain a reference to it. It is *not* safe to delete the **DDS::TypeCode** (p. 1301) 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::DynamicDataTypeSupport::create_data** (p. 826), and the **DDS::TypeCode** (p. 1301) 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::DynamicDataTypeSupport** (p. 822) object maintains the **DDS::TypeCode** (p. 1301) used by the samples it creates.

However you create a **DDS::DynamicData** (p. 719) 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.

In C#:

```
DynamicData sample = new DynamicData(
     myType, myProperties);
 // Do something...
 sample.dispose();
```

In C++/CLI:

```
DynamicData^ sample = gcnew DynamicData(
     myType, myProperties);
 // Do something...
 delete sample;
```

Parameters:

type *<<in>>* *(p. 175)* The type of which the new object will represent an object.

property *<<in>>* *(p. 175)* Properties that configure the behavior of the new object. Most users can simply use DDS::DynamicDataProperty_t::DYNAMIC_DATA_PROPERTY_DEFAULT.

See also:

DDS::DynamicData::delete
DDS::DynamicDataTypeSupport::create_data (p. 826)

6.54.5.2 DDS::DynamicData::~DynamicData ()

Finalize and deallocate this **DDS::DynamicData** (p. 719) sample.

MT Safety:

UNSAFE.

See also:

DDS::DynamicData::DynamicData (p. 734)

6.54.6 Member Function Documentation

6.54.6.1 virtual System::Boolean DDS::DynamicData::copy_from (DynamicData^ *other*) [virtual]

Deeply copy from the given object to this object.

MT Safety:

UNSAFE.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_OutOfResources** (p. 1122)

See also:

DDS::ICopyable (p. 902)

6.54.6.2 System::Boolean DDS::DynamicData::is_valid ()

Indicates whether the object was constructed properly.

This method returns true if the constructor succeeded; it returns false if the constructor failed for any reason, which should also have resulted in a log message. It is only necessary to call this method if you created the **DDS::DynamicData** (p. 719) object using the constructor, **DDS::DynamicData::DynamicData** (p. 734).

Possible failure reasons include passing an invalid type or invalid properties to the constructor.

This method is necessary because C++ exception support is not consistent across all of the platforms on which RTI Data Distribution Service runs. Therefore, the implementation does not throw any exceptions in the constructor.

MT Safety:

UNSAFE.

See also:

DDS::DynamicData::DynamicData (p. 734)

6.54.6.3 void DDS::DynamicData::copy (DynamicData^ *src*)

Deeply copy from the given object to this object.

MT Safety:

UNSAFE.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_OutOfResources** (p. 1122)

6.54.6.4 System::Boolean DDS::DynamicData::equal (DynamicData^ *other*)

Indicate whether the contents of another **DDS::DynamicData** (p. 719) sample are the same as those of this one.

This operation compares the data and type of existing members. The types of non-instantiated members may differ in sparse types.

MT Safety:

UNSAFE.

See also:

DDS::TCKind::TK_SPARSE

6.54.6.5 void DDS::DynamicData::clear_all_members ()

Clear the contents of all data members of this object, including key members.

MT Safety:

UNSAFE.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::clear_nonkey_members (p. 737)
DDS::DynamicData::clear_member (p. 737)

6.54.6.6 void DDS::DynamicData::clear_nonkey_members ()

Clear the contents of all data members of this object, not including key members.

This method is only applicable to sparse value types.

MT Safety:

UNSAFE.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::TCKind::TK_SPARSE
DDS::DynamicData::clear_all_members (p. 737)
DDS::DynamicData::clear_member (p. 737)

6.54.6.7 void DDS::DynamicData::clear_member (System::String^ *member_name*, System::Int32 *member_id*)

Clear the contents of a single data member of this object.

This method is only applicable to sparse value types.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::TCKind::TK_SPARSE
DDS::DynamicData::clear_all_members (p. 737)
DDS::DynamicData::clear_nonkey_members (p. 737)

6.54.6.8 void DDS::DynamicData::get_info (DynamicDataInfo^ *info_out*)

Fill in the given descriptor with information about this **DDS::DynamicData** (p. 719).

MT Safety:

UNSAFE.

Parameters:

info_out *<<out>>* *(p. 176)* The descriptor object whose contents will be overwritten by this operation.

6.54.6.9 void DDS::DynamicData::bind_type (DDS::TypeCode^ *type*)

If this **DDS::DynamicData** (p. 719) object is not yet associated with a data type, set that type now to the given **DDS::TypeCode** (p. 1301).

This advanced operation allows you to reuse a single **DDS::DynamicData** (p. 719) object with multiple data types.

In C#:

```
DynamicData myData = new DynamicData(null, myProperties);
 TypeCode myType = ...;
```

```
myData.bind_type(myType);
try {
    // Do something...
} finally {
    myData.unbind_type();
}
myData.Dispose()
```

In C++/CLI:

```
DynamicData^ myData = gcnew DynamicData(nullptr, myProperties);
TypeCode^ myType = ...;
myData->bind_type(myType);
try {
    // Do something...
} finally {
    myData->unbind_type();
}
delete myData;
```

Note that the **DDS::DynamicData** (p. 719) object will retain a reference to the **DDS::TypeCode** (p. 1301) object you provide. It is *not* safe to delete the **DDS::TypeCode** (p. 1301) until after it is unbound.

MT Safety:

UNSAFE.

Parameters:

type <<*in*>> *(p. 175)* The type to associate with this **DDS::DynamicData** (p. 719) object.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::unbind_type (p. 739)

6.54.6.10 void DDS::DynamicData::unbind_type ()

Dissociate this **DDS::DynamicData** (p. 719) 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.

MT Safety:

UNSAFE.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::bind_type (p. 738)
DDS::DynamicData::clear_all_members (p. 737)

6.54.6.11 void DDS::DynamicData::bind_complex_member (DynamicData^ *value_out*, System::String^ *member_name*, System::Int32 *member_id*)

Use another **DDS::DynamicData** (p. 719) object to provide access to a complex field of this **DDS::DynamicData** (p. 719) object.

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. 719) object to that member. This operation will bind the type code of the member to the provided **DDS::DynamicData** (p. 719) object and perform additional initialization.

The following example demonstrates the usage pattern. Note that error handling has been omitted for brevity.

In C#:

```
DynamicData outer = ...;
DynamicData toBeBound = new DynamicData(null, myProperties);
outer.bind_complex_member(
    toBeBound,
    "complexMember",
    DynamicData.MEMBER_ID_UNSPECIFIED);
try {
    float theFloatValue = toBeBound.get_float(
        "theFloat"
```

```
        DynamicData.MEMBER_ID_UNSPECIFIED);
} finally {
    outer.unbind_complex_member(toBeBound);
}
toBeBound.Dispose();
```

In C++/CLI:

```
DynamicData^ outer = ...;
DynamicData^ toBeBound = gcnew DynamicData(nullptr, myProperties);
outer->bind_complex_member(
    toBeBound,
    "complexMember",
    DynamicData::MEMBER_ID_UNSPECIFIED);
try {
    float theFloatValue = toBeBound->get_float(
        "theFloat"
        DynamicData::MEMBER_ID_UNSPECIFIED);
} finally {
    outer->unbind_complex_member(toBeBound);
}
delete toBeBound;
```

This operation is only permitted when the object `toBeBound` (named as in the example above) is not currently associated with any type, including already being bound to another member. You can see in the example that this object is created directly with the constructer and is not provided with a **DDS::TypeCode** (p. 1301).

Only a single member of a given **DDS::DynamicData** (p. 719) 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. 742) 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. 757) 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. 757) 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. 731).

MT Safety:

UNSAFE.

Parameters:

value_out *<<**out**>>* *(p. 176)* The object that you wish to bind to the field.

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::unbind_complex_member (p. 742)
DDS::DynamicData::get_complex_member (p. 757)

6.54.6.12 void DDS::DynamicData::unbind_complex_member (DynamicData^ *value*)

Tear down the association created by a **DDS::DynamicData::bind_complex_member** (p. 740) 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. 740) on a data sample, you must unbind before writing or deleting the sample.

MT Safety:

UNSAFE.

Parameters:

value *<<**in**>>* *(p. 175)* The same object you passed to **DDS::DynamicData::bind_complex_member** (p. 740). This argument is used for error checking purposes.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_OutOfResources** (p. 1122)

See also:

DDS::DynamicData::bind_complex_member (p. 740)

6.54.6.13 DDS::TypeCode ^ DDS::DynamicData::get_type ()

Get the data type, of which this **DDS::DynamicData** (p. 719) represents an instance.

MT Safety:

UNSAFE.

6.54.6.14 TCKind DDS::DynamicData::get_type_kind ()

Get the kind of this object's data type.

This is a convenience method. It's equivalent to calling **DDS::DynamicData::get_type** (p. 743) followed by **DDS::TypeCode::kind** (p. 1305).

MT Safety:

UNSAFE.

6.54.6.15 System::UInt32 DDS::DynamicData::get_member_count ()

Get the number of members in this sample.

For objects of type kind DDS::TCKind::TK_ARRAY or DDS::TCKind::TK_SEQUENCE, this method returns the number of elements in the collection.

For objects of type kind DDS::TCKind::TK_STRUCT or DDS::TCKind::TK_VALUE, it returns the number of fields in the sample, which will always be the same as the number of fields in the type.

For objects of type kind DDS::TCKind::TK_SPARSE, it returns the number of fields in the sample, which may be less than or equal to the number of fields in the type.

MT Safety:

UNSAFE.

See also:

DDS::DynamicData::get_member_info_by_index (p. 746)

6.54.6.16 System::Boolean DDS::DynamicData::member_exists (System::String^ *member_name*, System::Int32 *member_id*)

Indicates whether a member of a particular name/ID exists in this data sample.

Only one of the name and/or ID need by specified.

For objects of type kinds other than DDS::TCKind::TK_SPARSE, the result of this method will always be the same as that of **DDS::DynamicData::member_exists_in_type** (p. 744).

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

See also:

DDS::DynamicData::member_exists_in_type (p. 744)
DDS::DynamicData::MEMBER_ID_UNSPECIFIED (p. 75)

6.54.6.17 System::Boolean DDS::DynamicData::member_exists_in_type (System::String^ *member_name*, System::Int32 *member_id*)

Indicates whether a member of a particular name/ID exists in this data sample's type.

Only one of the name and/or ID need by specified.

For objects of type kinds other than DDS::TCKind::TK_SPARSE, the result of this method will always be the same as that of **DDS::DynamicData::member_exists** (p. 744).

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

See also:

DDS::DynamicData::member_exists (p. 744)
DDS::DynamicData::MEMBER_ID_UNSPECIFIED (p. 75)

6.54.6.18 void DDS::DynamicData::get_member_info (DynamicDataMemberInfo^ *info*, System::String^ *member_name*, System::Int32 *member_id*)

Fill in the given descriptor with information about the identified member of this **DDS::DynamicData** (p. 719) sample.

This operation is valid for objects of DDS::TCKind DDS::TCKind::TK_ARRAY, DDS::TCKind::TK_SEQUENCE, DDS::TCKind::TK_STRUCT, DDS::TCKind::TK_VALUE, and DDS::TCKind::TK_SPARSE.

MT Safety:

UNSAFE.

Parameters:

info *<<out>>* *(p. 176)* The descriptor object whose contents will be overwritten by this operations.

member_name *<<in>>* *(p. 175)* The name of the member for which to get the info or null to look up the member by its ID. Only one of the name and the ID may be unspecified.

member_id *<<in>>* *(p. 175)* The ID of the member for which to get the info, or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::get_member_info_by_index (p. 746)
DDS::DynamicData::MEMBER_ID_UNSPECIFIED (p. 75)

6.54.6.19 void DDS::DynamicData::get_member_info_by_index (DynamicDataMemberInfo^ *info*, System::UInt32 *index*)

Fill in the given descriptor with information about the identified member of this **DDS::DynamicData** (p. 719) sample.

This operation is valid for objects of DDS::TCKind DDS::TCKind::TK_ARRAY, DDS::TCKind::TK_SEQUENCE, DDS::TCKind::TK_STRUCT, DDS::TCKind::TK_VALUE, and DDS::TCKind::TK_SPARSE.

MT Safety:

UNSAFE.

Parameters:

info <<***out***>> *(p. 176)* The descriptor object whose contents will be overwritten by this operations.

index <<***in***>> *(p. 175)* The zero-based of the member for which to get the info.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::get_member_info (p. 745)
DDS::DynamicData::get_member_count (p. 743)
DDS::DynamicData::MEMBER_ID_UNSPECIFIED (p. 75)

6.54.6.20 void DDS::DynamicData::get_member_type (DDS::TypeCode^ % *type_out*, System::String^ *member_name*, System::Int32 *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 DDS::TCKind::TK_ARRAY, DDS::TCKind::TK_SEQUENCE, DDS::TCKind::TK_STRUCT, DDS::TCKind::TK_VALUE, and DDS::TCKind::TK_SPARSE. For type kinds DDS::TCKind::TK_ARRAY and DDS::TCKind::TK_SEQUENCE, 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.

MT Safety:

UNSAFE.

Parameters:

type_out *<<**out**>>* *(p. 176)* If this method returned success, this argument refers to the found member's type.

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::get_member_info (p. 745)
DDS::DynamicData::MEMBER_ID_UNSPECIFIED (p. 75)

6.54.6.21 System::Boolean DDS::DynamicData::is_member_key (System::String^ *member_name*, System::Int32 *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::TCKind::TK_STRUCT, DDS::TCKind::TK_VALUE, or DDS::TCKind::TK_SPARSE.

Note to users of sparse types: A key member may only have a single representation and is required to exist in every sample.

MT Safety:

UNSAFE.

Parameters:

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.54.6.22 System::Int32 DDS::DynamicData::get_int (System::String^ *member_name*, System::Int32 *member_id*)

Get the value of the given field, which is of type System::Int32 or another type implicitly convertible to it.

The member may be specified by name or by ID.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_int (p. 776)

6.54.6.23 System::Int16 DDS::DynamicData::get_short (System::String^ *member_name*, System::Int32 *member_id*)

Get the value of the given field, which is of type System::Int16 or another type implicitly convertible to it.

The member may be specified by name or by ID.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_short (p. 777)

6.54.6.24 System::UInt32 DDS::DynamicData::get_uint (System::String^ *member_name*, System::Int32 *member_id*)

Get the value of the given field, which is of type System::UInt32 or another type implicitly convertible to it.

The member may be specified by name or by ID.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_uint (p. 777)

6.54.6.25 System::UInt16 DDS::DynamicData::get_ushort (System::String^ *member_name*, System::Int32 *member_id*)

Get the value of the given field, which is of type System::UInt16 or another type implicitly convertible to it.

The member may be specified by name or by ID.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_ushort (p. 778)

6.54.6.26 System::Single DDS::DynamicData::get_float (System::String^ *member_name*, System::Int32 *member_id*)

Get the value of the given field, which is of type System::Single or another type implicitly convertible to it.

The member may be specified by name or by ID.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_float (p. 779)

6.54.6.27 System::Double DDS::DynamicData::get_double (System::String^ *member_name*, System::Int32 *member_id*)

Get the value of the given field, which is of type System::Double or another type implicitly convertible to it.

The member may be specified by name or by ID.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_double (p. 779)

6.54.6.28 System::Boolean DDS::DynamicData::get_boolean (System::String^ *member_name*, System::Int32 *member_id*)

Get the value of the given field, which is of type System::Boolean.

The member may be specified by name or by ID.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_boolean (p. 780)

6.54.6.29 System::Char DDS::DynamicData::get_char (System::String^ *member_name*, System::Int32 *member_id*)

Get the value of the given field, which is of type System::Char or another type implicitly convertible to it.

The member may be specified by name or by ID.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_char (p. 780)

6.54.6.30 System::Byte DDS::DynamicData::get_byte (System::String^ *member_name*, System::Int32 *member_id*)

Get the value of the given field, which is of type System::Byte or another type implicitly convertible to it.

The member may be specified by name or by ID.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_byte (p. 781)

6.54.6.31 System::Int64 DDS::DynamicData::get_long (System::String^ *member_name*, System::Int32 *member_id*)

Get the value of the given field, which is of type System::Int64 or another type implicitly convertible to it.

The member may be specified by name or by ID.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_long (p. 782)

6.54.6.32 System::UInt64 DDS::DynamicData::get_ulong (System::String^ *member_name*, System::Int32 *member_id*)

Get the value of the given field, which is of type System::UInt64 or another type implicitly convertible to it.

The member may be specified by name or by ID.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_ulong (p. 782)

6.54.6.33 LongDouble DDS::DynamicData::get_longdouble (System::String^ *member_name*, System::Int32 *member_id*)

Get the value of the given field, which is of type **DDS::LongDouble** (p. 976) or another type implicitly convertible to it.

The member may be specified by name or by ID.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_longdouble (p. 783)

6.54.6.34 System::Char DDS::DynamicData::get_wchar (System::String^ *member_name*, System::Int32 *member_id*)

Get the value of the given field, which is of type System::Char or another type implicitly convertible to it.

The member may be specified by name or by ID.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_wchar (p. 783)

6.54.6.35 System::String ^ DDS::DynamicData::get_string (System::String^ *member_name*, System::Int32 *member_id*)

Get the value of the given field, which is of type System::String.

The member may be specified by name or by ID.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::set_string (p. 784)

6.54.6.36 System::String ^ DDS::DynamicData::get_wstring (System::String^ *member_name*, System::Int32 *member_id*)

Get the value of the given field, which is of type System::String.

The member may be specified by name or by ID.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::set_wstring (p. 785)

6.54.6.37 void DDS::DynamicData::get_complex_member (DynamicData^ *value_out*, System::String^ *member_name*, System::Int32 *member_id*)

Get a copy of the value of the given field, which is of some composed type.

The member may be of type kind DDS::TCKind::TK_ARRAY, DDS::TCKind::TK_SEQUENCE, DDS::TCKind::TK_STRUCT, DDS::TCKind::TK_VALUE, DDS::TCKind::TK_UNION, or DDS::TCKind::TK_SPARSE. It may be specified by name or by ID.

This method is logically related to **DDS::DynamicData::bind_complex_member** (p. 740) in that both allow you to examine the state of nested objects. They are different in an important way: this method provides a *copy* of the data; changes to it will not be reflected in the source object.

MT Safety:

UNSAFE.

Parameters:

value_out *<<out>> (p. 176)* The **DDS::DynamicData** (p. 719) sample whose contents will be overwritten by this operation. This object must *not* be a bound member of another **DDS::DynamicData** (p. 719) sample.

member_name *<<in>> (p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>> (p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_OutOfResources** (p. 1122)

See also:

DDS::DynamicData::set_complex_member (p. 785)
DDS::DynamicData::bind_complex_member (p. 740)

6.54.6.38 void DDS::DynamicData::get_int_array (array< System::Int32 >^ *array*, System::String^ *member_name*, System::Int32 *member_id*)

Get a copy of the given array member.

This method will perform an automatic conversion from IntSeq.

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

array *<<**out**>>* *(p. 176)* An already-allocated array, into which the elements will be copied.

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_int_array (p. 787)
DDS::DynamicData::get_int_seq (p. 767)

6.54.6.39 void DDS::DynamicData::get_short_array (array< System::Int16 >^ *array*, System::String^ *member_name*, System::Int32 *member_id*)

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS::ShortSeq** (p. 1181).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

array *<<**out**>>* *(p. 176)* An already-allocated array, into which the elements will be copied.

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_short_array (p. 788)
DDS::DynamicData::get_short_seq (p. 768)

6.54.6.40 void DDS::DynamicData::get_uint_array (array< System::UInt32 >^ *array*, System::String^ *member_name*, System::Int32 *member_id*)

Get a copy of the given array member.

This method will perform an automatic conversion from UnsignedIntSeq.

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

array *<<out>>* *(p. 176)* An already-allocated array, into which the elements will be copied.

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_uint (p. 777)
DDS::DynamicData::get_uint_seq (p. 768)

6.54.6.41 void DDS::DynamicData::get_ushort_array (array< System::UInt16 >^ *array*, System::String^ *member_name*, System::Int32 *member_id*)

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS::UnsignedShortSeq** (p. 1401).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

array *<<**out**>>* *(p. 176)* An already-allocated array, into which the elements will be copied.

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_ushort_array (p. 789)
DDS::DynamicData::get_ushort_seq (p. 769)

6.54.6.42 void DDS::DynamicData::get_float_array (array< System::Single >^ *array*, System::String^ *member_name*, System::Int32 *member_id*)

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS::FloatSeq** (p. 865).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

array *<<**out**>>* *(p. 176)* An already-allocated array, into which the elements will be copied.

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_float_array (p. 790)
DDS::DynamicData::get_float_seq (p. 770)

6.54.6.43 void DDS::DynamicData::get_double_array (array< System::Double >^ *array*, System::String^ *member_name*, System::Int32 *member_id*)

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS::DoubleSeq** (p. 707).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

array *<<**out**>>* *(p. 176)* An already-allocated array, into which the elements will be copied.

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_double_array (p. 791)
DDS::DynamicData::get_double_seq (p. 770)

6.54.6.44 void DDS::DynamicData::get_boolean_array (array< System::Boolean >^ *array*, System::String^ *member_name*, System::Int32 *member_id*)

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS::BooleanSeq** (p. 381).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

array *<<**out**>>* *(p. 176)* An already-allocated array, into which the elements will be copied.

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_boolean_array (p. 791)
DDS::DynamicData::get_boolean_seq (p. 771)

6.54.6.45 void DDS::DynamicData::get_char_array (array< System::Char >^ *array*, System::String^ *member_name*, System::Int32 *member_id*)

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS::CharSeq** (p. 406).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

array *<<**out**>>* *(p. 176)* An already-allocated array, into which the elements will be copied.

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_char_array (p. 792)
DDS::DynamicData::get_char_seq (p. 772)

6.54.6.46 void DDS::DynamicData::get_byte_array (array< System::Byte >^ *array*, System::String^ *member_name*, System::Int32 *member_id*)

Get a copy of the given array member.

This method will perform an automatic conversion from ByteSeq.

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

array *<<**out**>>* *(p. 176)* An already-allocated array, into which the elements will be copied.

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_byte_array (p. 793)
DDS::DynamicData::get_byte_seq (p. 773)

6.54.6.47 void DDS::DynamicData::get_long_array (array< System::Int64 >^ *array*, System::String^ *member_name*, System::Int32 *member_id*)

Get a copy of the given array member.

This method will perform an automatic conversion from LongSeq.

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

array *<<**out**>>* *(p. 176)* An already-allocated array, into which the elements will be copied.

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_long_array (p. 794)
DDS::DynamicData::get_long_seq (p. 773)

6.54.6.48 void DDS::DynamicData::get_ulong_array (array< System::UInt64 >^ *array*, System::String^ *member_name*, System::Int32 *member_id*)

Get a copy of the given array member.

This method will perform an automatic conversion from DDS::UnsignedLongLongSeq.

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

array <<*out*>> *(p. 176)* An already-allocated array, into which the elements will be copied.

member_name <<*in*>> *(p. 175)* The name of the member or null to look up the member by its ID.

member_id <<*in*>> *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_ulong_array (p. 794)
DDS::DynamicData::get_ulong_seq (p. 774)

6.54.6.49 void DDS::DynamicData::get_longdouble_array (array< LongDouble >^ *array*, System::String^ *member_name*, System::Int32 *member_id*)

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS::LongDoubleSeq** (p. 977).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

array *<<**out**>>* *(p. 176)* An already-allocated array, into which the elements will be copied.

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_longdouble_array (p. 795)
DDS::DynamicData::get_longdouble_seq (p. 775)

6.54.6.50 void DDS::DynamicData::get_wchar_array (array< System::Char >^ *array*, System::String^ *member_name*, System::Int32 *member_id*)

Get a copy of the given array member.

This method will perform an automatic conversion from **DDS::WcharSeq** (p. 1421).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

array *<<**out**>>* *(p. 176)* An already-allocated array, into which the elements will be copied.

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::DynamicData::set_wchar_array (p. 796)
DDS::DynamicData::get_wchar_seq (p. 775)

6.54.6.51 void DDS::DynamicData::get_int_seq (IntSeq^ *seq*, System::String^ *member_name*, System::Int32 *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 System::Int32.

MT Safety:

UNSAFE.

Parameters:

seq *<<out>>* *(p. 176)* A sequence, into which the elements will be copied.

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_OutOfResources** (p. 1122)

See also:

DDS::DynamicData::set_int_seq (p. 796)
DDS::DynamicData::get_int_array (p. 757)

6.54.6.52 void DDS::DynamicData::get_short_seq (ShortSeq^ *seq*, System::String^ *member_name*, System::Int32 *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 System::Int16.

MT Safety:

UNSAFE.

Parameters:

seq *<<**out**>>* *(p. 176)* A sequence, into which the elements will be copied.

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_OutOfResources** (p. 1122)

See also:

DDS::DynamicData::set_short_seq (p. 797)
DDS::DynamicData::get_short_array (p. 758)

6.54.6.53 void DDS::DynamicData::get_uint_seq (UnsignedIntSeq^ *seq*, System::String^ *member_name*, System::Int32 *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 System::UInt32.

MT Safety:

UNSAFE.

Parameters:

seq *<<**out**>>* *(p. 176)* A sequence, into which the elements will be copied.

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::set_uint_seq (p. 798)
DDS::DynamicData::get_uint_array (p. 759)

6.54.6.54 void DDS::DynamicData::get_ushort_seq (UnsignedShortSeq^ *seq*, System::String^ *member_name*, System::Int32 *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 System::UInt16.

MT Safety:

UNSAFE.

Parameters:

seq *<<**out**>>* *(p. 176)* A sequence, into which the elements will be copied.

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::set_ushort_seq (p. 799)
DDS::DynamicData::get_ushort_array (p. 760)

6.54.6.55 void DDS::DynamicData::get_float_seq (FloatSeq^ *seq*, System::String^ *member_name*, System::Int32 *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 System::Single.

MT Safety:

UNSAFE.

Parameters:

seq *<<**out**>>* *(p. 176)* A sequence, into which the elements will be copied.

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_OutOfResources** (p. 1122)

See also:

DDS::DynamicData::set_float_seq (p. 799)
DDS::DynamicData::get_float_array (p. 760)

6.54.6.56 void DDS::DynamicData::get_double_seq (DoubleSeq^ *seq*, System::String^ *member_name*, System::Int32 *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 System::Double.

MT Safety:

UNSAFE.

Parameters:

seq *<<**out**>>* *(p. 176)* A sequence, into which the elements will be copied.

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_OutOfResources** (p. 1122)

See also:

DDS::DynamicData::set_double_seq (p. 800)
DDS::DynamicData::get_double_array (p. 761)

6.54.6.57 void DDS::DynamicData::get_boolean_seq (BooleanSeq^ *seq*, System::String^ *member_name*, System::Int32 *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 System::Boolean.

MT Safety:

UNSAFE.

Parameters:

seq *<<**out**>>* *(p. 176)* A sequence, into which the elements will be copied.

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::set_boolean_seq (p. 801)
DDS::DynamicData::get_boolean_array (p. 762)

6.54.6.58 void DDS::DynamicData::get_char_seq (CharSeq^ *seq*, System::String^ *member_name*, System::Int32 *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 System::Char.

MT Safety:

UNSAFE.

Parameters:

seq *<<out>>* *(p. 176)* A sequence, into which the elements will be copied.

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::set_char_seq (p. 801)
DDS::DynamicData::get_char_array (p. 763)

6.54.6.59 void DDS::DynamicData::get_byte_seq (ByteSeq^ *seq*, System::String^ *member_name*, System::Int32 *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 System::Byte.

MT Safety:

UNSAFE.

Parameters:

seq *<<**out**>>* *(p. 176)* A sequence, into which the elements will be copied.

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::set_byte_seq (p. 802)
DDS::DynamicData::get_byte_array (p. 763)

6.54.6.60 void DDS::DynamicData::get_long_seq (LongSeq^ *seq*, System::String^ *member_name*, System::Int32 *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 System::Int64.

MT Safety:

UNSAFE.

Parameters:

seq *<<**out**>>* *(p. 176)* A sequence, into which the elements will be copied.

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::set_long_seq (p. 803)
DDS::DynamicData::get_long_array (p. 764)

6.54.6.61 void DDS::DynamicData::get_ulong_seq (UnsignedLongSeq^ *seq*, System::String^ *member_name*, System::Int32 *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 System::UInt64.

MT Safety:

UNSAFE.

Parameters:

seq *<<**out**>>* *(p. 176)* A sequence, into which the elements will be copied.

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::set_ulong_seq (p. 803)
DDS::DynamicData::get_ulong_array (p. 765)

6.54.6.62 void DDS::DynamicData::get_longdouble_seq (LongDoubleSeq^ *seq*, System::String^ *member_name*, System::Int32 *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. 976).

MT Safety:

UNSAFE.

Parameters:

seq *<<**out**>>* *(p. 176)* A sequence, into which the elements will be copied.

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::set_longdouble_seq (p. 804)
DDS::DynamicData::get_longdouble_array (p. 765)

6.54.6.63 void DDS::DynamicData::get_wchar_seq (WcharSeq^ *seq*, System::String^ *member_name*, System::Int32 *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 System::Char.

MT Safety:

UNSAFE.

Parameters:

seq *<<**out**>>* *(p. 176)* A sequence, into which the elements will be copied.

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::set_wchar_seq (p. 805)
DDS::DynamicData::get_wchar_array (p. 766)

6.54.6.64 void DDS::DynamicData::set_int (System::String^ *member_name*, System::Int32 *member_id*, System::Int32 *value*)

Set the value of the given field, which is of type System::Int32.

MT Safety:

UNSAFE.

Parameters:

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<**in**>>* *(p. 175)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_int (p. 748)

6.54.6.65 void DDS::DynamicData::set_short (System::String^ *member_name*, System::Int32 *member_id*, System::Int16 *value*)

Set the value of the given field, which is of type System::Int16.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<in>>* *(p. 175)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_short (p. 748)

6.54.6.66 void DDS::DynamicData::set_uint (System::String^ *member_name*, System::Int32 *member_id*, System::UInt32 *value*)

Set the value of the given field, which is of type System::UInt32.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<in>>* *(p. 175)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_uint (p. 749)

6.54.6.67 void DDS::DynamicData::set_ushort (System::String^ *member_name*, System::Int32 *member_id*, System::UInt16 *value*)

Set the value of the given field, which is of type System::UInt16.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<in>>* *(p. 175)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_ushort (p. 749)

6.54.6.68 void DDS::DynamicData::set_float (System::String^ *member_name*, System::Int32 *member_id*, System::Single *value*)

Set the value of the given field, which is of type System::Single.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<in>>* *(p. 175)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_float (p. 750)

6.54.6.69 void DDS::DynamicData::set_double (System::String^ *member_name*, System::Int32 *member_id*, System::Double *value*)

Set the value of the given field, which is of type System::Double.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<in>>* *(p. 175)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_double (p. 751)

6.54.6.70 void DDS::DynamicData::set_boolean (System::String^ *member_name*, System::Int32 *member_id*, System::Boolean *value*)

Set the value of the given field, which is of type System::Boolean.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<in>>* *(p. 175)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_boolean (p. 751)

6.54.6.71 void DDS::DynamicData::set_char (System::String^ *member_name*, System::Int32 *member_id*, System::Char *value*)

Set the value of the given field, which is of type System::Char.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<in>>* *(p. 175)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_char (p. 752)

6.54.6.72 void DDS::DynamicData::set_byte (System::String^ *member_name*, System::Int32 *member_id*, System::Byte *value*)

Set the value of the given field, which is of type System::Byte.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<in>>* *(p. 175)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_byte (p. 752)

6.54.6.73 void DDS::DynamicData::set_long (System::String^ *member_name*, System::Int32 *member_id*, System::Int64 *value*)

Set the value of the given field, which is of type System::Int64.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<in>>* *(p. 175)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_long (p. 753)

6.54.6.74 void DDS::DynamicData::set_ulong (System::String^ *member_name*, System::Int32 *member_id*, System::UInt64 *value*)

Set the value of the given field, which is of type System::UInt64.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<in>>* *(p. 175)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_ulong (p. 754)

6.54.6.75 void DDS::DynamicData::set_longdouble (System::String^ *member_name*, System::Int32 *member_id*, LongDouble *value*)

Set the value of the given field, which is of type **DDS::LongDouble** (p. 976).

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<in>>* *(p. 175)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_longdouble (p. 754)

6.54.6.76 void DDS::DynamicData::set_wchar (System::String^ *member_name*, System::Int32 *member_id*, System::Char *value*)

Set the value of the given field, which is of type System::Char.

MT Safety:

UNSAFE.

Parameters:

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<**in**>>* *(p. 175)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_char (p. 752)

6.54.6.77 void DDS::DynamicData::set_string (System::String^ *member_name*, System::Int32 *member_id*, System::String^ *value*)

Set the value of the given field of type System::String.

MT Safety:

UNSAFE.

Parameters:

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<**in**>>* *(p. 175)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_string (p. 755)

6.54.6.78 void DDS::DynamicData::set_wstring (System::String^ *member_name*, System::Int32 *member_id*, System::String^ *value*)

Set the value of the given field of type System::String.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<in>>* *(p. 175)* The value to which to set the member.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_wstring (p. 756)

6.54.6.79 void DDS::DynamicData::set_complex_member (System::String^ *member_name*, System::Int32 *member_id*, DynamicData^ *value*)

Copy the state of the given **DDS::DynamicData** (p. 719) object into a member of this object.

The member may be of type kind DDS::TCKind::TK_-ARRAY, DDS::TCKind::TK_SEQUENCE, DDS::TCKind::TK_-STRUCT, DDS::TCKind::TK_VALUE, DDS::TCKind::TK_UNION, or DDS::TCKind::TK_SPARSE. It may be specified by name or by ID.

Example: Copying Data

This method can be used with **DDS::DynamicData::bind_complex_-member** (p. 740) to copy from one **DDS::DynamicData** (p. 719) object to another efficiently. Suppose the following data structure:

```
struct Bar {
     short theShort;
 };

 struct Foo {
     Bar theBar;
 };
```

Support we have two instances of `Foo` (p. 877): `foo_dst` and `foo_src`. We want to replace the contents of `foo_dst.theBar` with the contents of `foo_src.theBar`. Error handling has been omitted for the sake of brevity.

In C#:

```
DynamicData foo_dst = ...;
 DynamicData foo_src = ...;
 DynamicData bar = new DynamicData(null, myProperties);
 // Point to the source of the copy:
 foo_src.bind_complex_member(
     "theBar",
     DynamicData.MEMBER_ID_UNSPECIFIED,
     bar);
 try {
     // Just one copy:
     foo_dst.set_complex_member(
         "theBar",
         DynamicData.MEMBER_ID_UNSPECIFIED,
         bar);
 } finally {
     // Tear down:
     foo_src.unbind_complex_member(bar);
 }
 bar.Dispose();
```

In C++/CLI:

```
DynamicData^ foo_dst = ...;
 DynamicData^ foo_src = ...;
 DynamicData^ bar = gcnew DynamicData(nullptr, myProperties);
 // Point to the source of the copy:
 foo_src->bind_complex_member(
     "theBar",
     DynamicData::MEMBER_ID_UNSPECIFIED,
     bar);
 try {
     // Just one copy:
     foo_dst->set_complex_member(
         "theBar",
         DynamicData::MEMBER_ID_UNSPECIFIED,
         bar);
 } finally {
     // Tear down:
     foo_src->unbind_complex_member(bar);
 }
 delete bar;
```

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<in>>* *(p. 175)* The source **DDS::DynamicData** (p. 719) object whose contents will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_complex_member (p. 757)
DDS::DynamicData::bind_complex_member (p. 740)

6.54.6.80 void DDS::DynamicData::set_int_array (System::String^ *member_name*, System::Int32 *member_id*, array< System::Int32 >^ *array*)

Set the contents of the given array member.

This method will perform an automatic conversion to IntSeq.

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

array *<<in>>* *(p. 175)* The elements to copy.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_int_array (p. 757)
DDS::DynamicData::set_int_seq (p. 796)

6.54.6.81 void DDS::DynamicData::set_short_array (System::String^ *member_name*, System::Int32 *member_id*, array< System::Int16 >^ *array*)

Set the contents of the given array member.

This method will perform an automatic conversion to **DDS::ShortSeq** (p. 1181).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>> (p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>> (p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

array *<<in>> (p. 175)* The elements to copy.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_short_array (p. 758)
DDS::DynamicData::set_short_seq (p. 797)

6.54.6.82 void DDS::DynamicData::set_uint_array (System::String^ *member_name*, System::Int32 *member_id*, array< System::UInt32 >^ *array*)

Set the contents of the given array member.

This method will perform an automatic conversion to UnsignedIntSeq.

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

array *<<in>>* *(p. 175)* The elements to copy.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_uint_array (p. 759)
DDS::DynamicData::set_uint_seq (p. 798)

6.54.6.83 void DDS::DynamicData::set_ushort_array (System::String^ *member_name*, System::Int32 *member_id*, array< System::UInt16 >^ *array*)

Set the contents of the given array member.

This method will perform an automatic conversion to **DDS::UnsignedShortSeq** (p. 1401).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

array *<<in>>* *(p. 175)* The elements to copy.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_ushort_array (p. 760)
DDS::DynamicData::set_ushort_seq (p. 799)

6.54.6.84 void DDS::DynamicData::set_float_array (System::String^ *member_name*, System::Int32 *member_id*, array< System::Single >^ *array*)

Set the contents of the given array member.

This method will perform an automatic conversion to **DDS::FloatSeq** (p. 865).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

array *<<in>>* *(p. 175)* The elements to copy.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_float_array (p. 760)
DDS::DynamicData::set_float_seq (p. 799)

6.54.6.85 void DDS::DynamicData::set_double_array (System::String^ *member_name*, System::Int32 *member_id*, array< System::Double >^ *array*)

Set the contents of the given array member.

This method will perform an automatic conversion to **DDS::DoubleSeq** (p. 707).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

array *<<in>>* *(p. 175)* The elements to copy.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_double_array (p. 761)
DDS::DynamicData::set_double_seq (p. 800)

6.54.6.86 void DDS::DynamicData::set_boolean_array (System::String^ *member_name*, System::Int32 *member_id*, array< System::Boolean >^ *array*)

Set the contents of the given array member.

This method will perform an automatic conversion to **DDS::BooleanSeq** (p. 381).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

array *<<in>>* *(p. 175)* The elements to copy.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_boolean_array (p. 762)
DDS::DynamicData::set_boolean_seq (p. 801)

6.54.6.87 void DDS::DynamicData::set_char_array (System::String^ *member_name*, System::Int32 *member_id*, array< System::Char >^ *array*)

Set the contents of the given array member.

This method will perform an automatic conversion to **DDS::CharSeq** (p. 406).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

array *<<in>>* *(p. 175)* The elements to copy.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_char_array (p. 763)
DDS::DynamicData::set_char_seq (p. 801)

6.54.6.88 void DDS::DynamicData::set_byte_array (System::String^ *member_name*, System::Int32 *member_id*, array< System::Byte >^ *array*)

Set the contents of the given array member.

This method will perform an automatic conversion to ByteSeq.

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

array *<<in>>* *(p. 175)* The elements to copy.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_byte_array (p. 763)
DDS::DynamicData::set_byte_seq (p. 802)

6.54.6.89 void DDS::DynamicData::set_long_array (System::String^ *member_name*, System::Int32 *member_id*, array< System::Int64 >^ *array*)

Set the contents of the given array member.

This method will perform an automatic conversion to LongSeq.

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

array *<<in>>* *(p. 175)* The elements to copy.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_long_array (p. 764)
DDS::DynamicData::set_long_seq (p. 803)

6.54.6.90 void DDS::DynamicData::set_ulong_array (System::String^ *member_name*, System::Int32 *member_id*, array< System::UInt64 >^ *array*)

Set the contents of the given array member.

This method will perform an automatic conversion to DDS::UnsignedLongLongSeq.

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

array *<<in>>* *(p. 175)* The elements to copy.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_ulong_array (p. 765)
DDS::DynamicData::set_ulong_seq (p. 803)

6.54.6.91 void DDS::DynamicData::set_longdouble_array (System::String^ *member_name*, System::Int32 *member_id*, array< LongDouble >^ *array*)

Set the contents of the given array member.

This method will perform an automatic conversion to **DDS::LongDoubleSeq** (p. 977).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

array *<<in>>* *(p. 175)* The elements to copy.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_longdouble_array (p. 765)
DDS::DynamicData::set_longdouble_seq (p. 804)

6.54.6.92 void DDS::DynamicData::set_wchar_array (System::String^ *member_name*, System::Int32 *member_id*, array< System::Char >^ *array*)

Set the contents of the given array member.

This method will perform an automatic conversion to **DDS::WcharSeq** (p. 1421).

If the destination array is insufficiently long to store the data, this operation will fail without copying anything.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* (p. 175) The name of the member or null to look up the member by its ID.

member_id *<<in>>* (p. 175) The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

array *<<in>>* (p. 175) The elements to copy.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_wchar_array (p. 766)
DDS::DynamicData::set_wchar_seq (p. 805)

6.54.6.93 void DDS::DynamicData::set_int_seq (System::String^ *member_name*, System::Int32 *member_id*, IntSeq^ *value*)

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of System::Int32.

MT Safety:

UNSAFE.

Parameters:

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<**out**>>* *(p. 176)* A sequence, from which the elements will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_int_seq (p. 767)
DDS::DynamicData::set_int_array (p. 787)

6.54.6.94 void DDS::DynamicData::set_short_seq (System::String^ *member_name*, System::Int32 *member_id*, ShortSeq^ *value*)

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of System::Int16.

MT Safety:

UNSAFE.

Parameters:

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<**out**>>* *(p. 176)* A sequence, from which the elements will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_short_seq (p. 768)
DDS::DynamicData::set_short_array (p. 788)

6.54.6.95 void DDS::DynamicData::set_uint_seq (System::String^ *member_name*, System::Int32 *member_id*, UnsignedIntSeq^ *value*)

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of System::UInt32.

MT Safety:

UNSAFE.

Parameters:

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<**out**>>* *(p. 176)* A sequence, from which the elements will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_uint_seq (p. 768)
DDS::DynamicData::set_uint (p. 777)

6.54.6.96 void DDS::DynamicData::set_ushort_seq (System::String^ *member_name*, System::Int32 *member_id*, UnsignedShortSeq^ *value*)

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of System::UInt16.

MT Safety:

UNSAFE.

Parameters:

member_name <<*in*>> *(p. 175)* The name of the member or null to look up the member by its ID.

member_id <<*in*>> *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value <<*out*>> *(p. 176)* A sequence, from which the elements will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_ushort_seq (p. 769)
DDS::DynamicData::set_ushort_array (p. 789)

6.54.6.97 void DDS::DynamicData::set_float_seq (System::String^ *member_name*, System::Int32 *member_id*, FloatSeq^ *value*)

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of System::Single.

MT Safety:

UNSAFE.

Parameters:

member_name <<*in*>> *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<out>>* *(p. 176)* A sequence, from which the elements will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_float_seq (p. 770)
DDS::DynamicData::set_float_array (p. 790)

6.54.6.98 void DDS::DynamicData::set_double_seq (System::String^ *member_name*, System::Int32 *member_id*, DoubleSeq^ *value*)

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of System::Double.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<out>>* *(p. 176)* A sequence, from which the elements will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_double_seq (p. 770)
DDS::DynamicData::set_double_array (p. 791)

6.54.6.99 void DDS::DynamicData::set_boolean_seq (System::String^ *member_name*, System::Int32 *member_id*, BooleanSeq^ *value*)

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of System::Boolean.

MT Safety:

UNSAFE.

Parameters:

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<**in**>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<**out**>>* *(p. 176)* A sequence, from which the elements will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_boolean_seq (p. 771)
DDS::DynamicData::set_boolean_array (p. 791)

6.54.6.100 void DDS::DynamicData::set_char_seq (System::String^ *member_name*, System::Int32 *member_id*, CharSeq^ *value*)

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of System::Char.

MT Safety:

UNSAFE.

Parameters:

member_name *<<**in**>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<out>>* *(p. 176)* A sequence, from which the elements will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_char_seq (p. 772)
DDS::DynamicData::set_char_array (p. 792)

6.54.6.101 void DDS::DynamicData::set_byte_seq (System::String^ *member_name*, System::Int32 *member_id*, ByteSeq^ *value*)

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of System::Byte.

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<out>>* *(p. 176)* A sequence, from which the elements will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_byte_seq (p. 773)
DDS::DynamicData::set_byte_array (p. 793)

6.54.6.102 void DDS::DynamicData::set_long_seq (System::String^ *member_name*, System::Int32 *member_id*, LongSeq^ *value*)

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of System::Int64.

MT Safety:

UNSAFE.

Parameters:

member_name <<*in*>> *(p. 175)* The name of the member or null to look up the member by its ID.

member_id <<*in*>> *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value <<*out*>> *(p. 176)* A sequence, from which the elements will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_long_seq (p. 773)
DDS::DynamicData::set_long_array (p. 794)

6.54.6.103 void DDS::DynamicData::set_ulong_seq (System::String^ *member_name*, System::Int32 *member_id*, UnsignedLongSeq^ *value*)

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of System::UInt64.

MT Safety:

UNSAFE.

Parameters:

member_name <<*in*>> *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<out>>* *(p. 176)* A sequence, from which the elements will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_ulong_seq (p. 774)
DDS::DynamicData::set_ulong_array (p. 794)

6.54.6.104 void DDS::DynamicData::set_longdouble_seq (System::String^ *member_name*, System::Int32 *member_id*, LongDoubleSeq^ *value*)

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of **DDS::LongDouble** (p. 976).

MT Safety:

UNSAFE.

Parameters:

member_name *<<in>>* *(p. 175)* The name of the member or null to look up the member by its ID.

member_id *<<in>>* *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value *<<out>>* *(p. 176)* A sequence, from which the elements will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_longdouble_seq (p. 775)
DDS::DynamicData::set_longdouble_array (p. 795)

6.54.6.105 void DDS::DynamicData::set_wchar_seq (System::String^ *member_name*, System::Int32 *member_id*, WcharSeq^ *value*)

Set the contents of the given sequence member.

This method will perform an automatic conversion to an array of System::Char.

MT Safety:

UNSAFE.

Parameters:

member_name <<*in*>> *(p. 175)* The name of the member or null to look up the member by its ID.

member_id <<*in*>> *(p. 175)* The ID of the member or **DDS::DynamicData::MEMBER_ID_UNSPECIFIED** (p. 75) to look up by name. See **Member Names and IDs** (p. 730).

value <<*out*>> *(p. 176)* A sequence, from which the elements will be copied.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_OutOfResources** (p. 1122)

See also:

DDS::DynamicData::get_wchar_seq (p. 775)
DDS::DynamicData::set_wchar_array (p. 796)

Get	Set
DDS::DynamicData::get_int (p. 748)	**DDS::DynamicData::set_int** (p. 776)
DDS::DynamicData::get_uint (p. 749)	**DDS::DynamicData::set_uint** (p. 777)
DDS::DynamicData::get_short (p. 748)	**DDS::DynamicData::set_short** (p. 777)
DDS::DynamicData::get_ushort (p. 749)	**DDS::DynamicData::set_ushort** (p. 778)
DDS::DynamicData::get_long (p. 753)	**DDS::DynamicData::set_long** (p. 782)
DDS::DynamicData::get_ulong (p. 754)	**DDS::DynamicData::set_ulong** (p. 782)
DDS::DynamicData::get_float (p. 750)	**DDS::DynamicData::set_float** (p. 779)
DDS::DynamicData::get_double (p. 751)	**DDS::DynamicData::set_double** (p. 779)
DDS::DynamicData::get_longdouble (p. 754)	**DDS::DynamicData::set_longdouble** (p. 783)
DDS::DynamicData::get_boolean (p. 751)	**DDS::DynamicData::set_boolean** (p. 780)
DDS::DynamicData::get_byte (p. 752)	**DDS::DynamicData::set_byte** (p. 781)
DDS::DynamicData::get_char (p. 752)	**DDS::DynamicData::set_char** (p. 780)
DDS::DynamicData::get_wchar (p. 755)	**DDS::DynamicData::set_wchar** (p. 783)
DDS::DynamicData::get_string (p. 755)	**DDS::DynamicData::set_string** (p. 784)
DDS::DynamicData::get_wstring (p. 756)	**DDS::DynamicData::set_wstring** (p. 785)

Table 6.2: Basic Types

Get	Set
DDS::DynamicData::get_complex_member (p. 757)	**DDS::DynamicData::set_complex_member** (p. 785)

Table 6.3: Structures, Arrays, and Other Complex Types

Get	Set
DDS::DynamicData::get_int_-array (p. 757)	**DDS::DynamicData::set_int_-array** (p. 787)
DDS::DynamicData::get_uint_-array (p. 759)	**DDS::DynamicData::set_uint** (p. 777)
DDS::DynamicData::get_short_-array (p. 758)	**DDS::DynamicData::set_short_-array** (p. 788)
DDS::DynamicData::get_-ushort_array (p. 760)	**DDS::DynamicData::set_-ushort_array** (p. 789)
DDS::DynamicData::get_long_-array (p. 764)	**DDS::DynamicData::set_long_-array** (p. 794)
DDS::DynamicData::get_-ulong_array (p. 765)	**DDS::DynamicData::set_ulong_-array** (p. 794)
DDS::DynamicData::get_float_-array (p. 760)	**DDS::DynamicData::set_float_-array** (p. 790)
DDS::DynamicData::get_-double_array (p. 761)	**DDS::DynamicData::set_-double_array** (p. 791)
DDS::DynamicData::get_-longdouble_array (p. 765)	**DDS::DynamicData::set_-longdouble_array** (p. 795)
DDS::DynamicData::get_-boolean_array (p. 762)	**DDS::DynamicData::set_-boolean** (p. 780)
DDS::DynamicData::get_byte_-array (p. 763)	**DDS::DynamicData::set_byte_-array** (p. 793)
DDS::DynamicData::get_char_-array (p. 763)	**DDS::DynamicData::set_char_-array** (p. 792)
DDS::DynamicData::get_-wchar_array (p. 766)	**DDS::DynamicData::set_-wchar_array** (p. 796)

Table 6.4: Arrays of Basic Types

Get	Set
DDS::DynamicData::get_int_seq (p. 767)	**DDS::DynamicData::set_int_seq** (p. 796)
DDS::DynamicData::get_uint_seq (p. 768)	**DDS::DynamicData::set_uint_seq** (p. 798)
DDS::DynamicData::get_short_seq (p. 768)	**DDS::DynamicData::set_short_seq** (p. 797)
DDS::DynamicData::get_ushort_seq (p. 769)	**DDS::DynamicData::set_ushort_seq** (p. 799)
DDS::DynamicData::get_long_seq (p. 773)	**DDS::DynamicData::set_long_seq** (p. 803)
DDS::DynamicData::get_ulong_seq (p. 774)	**DDS::DynamicData::set_ulong_seq** (p. 803)
DDS::DynamicData::get_float_seq (p. 770)	**DDS::DynamicData::set_float_seq** (p. 799)
DDS::DynamicData::get_double_seq (p. 770)	**DDS::DynamicData::set_double_seq** (p. 800)
DDS::DynamicData::get_longdouble_seq (p. 775)	**DDS::DynamicData::set_longdouble_seq** (p. 804)
DDS::DynamicData::get_boolean_seq (p. 771)	**DDS::DynamicData::set_boolean_seq** (p. 801)
DDS::DynamicData::get_byte_seq (p. 773)	**DDS::DynamicData::set_byte_seq** (p. 802)
DDS::DynamicData::get_char_seq (p. 772)	**DDS::DynamicData::set_char_seq** (p. 801)
DDS::DynamicData::get_wchar_seq (p. 775)	**DDS::DynamicData::set_wchar_seq** (p. 805)

Table 6.5: Sequences of Basic Types

6.55 DDS::DynamicDataInfo Class Reference

A descriptor for a **DDS::DynamicData** (p. 719) object.

`#include <managed_dynamicdata.h>`

Properties

- ^ System::Int32 **member_count** `[get]`

  *The number of data members in this **DDS::DynamicData** (p. 719) sample.*

- ^ System::Int32 **stored_size** `[get]`

  *The number of data members in this **DDS::DynamicData** (p. 719) sample.*

6.55.1 Detailed Description

A descriptor for a **DDS::DynamicData** (p. 719) object.

See also:

DDS::DynamicData::get_info (p. 738)

6.55.2 Property Documentation

6.55.2.1 System:: Int32 DDS::DynamicDataInfo::member_count `[get]`

The number of data members in this **DDS::DynamicData** (p. 719) sample.

6.55.2.2 System:: Int32 DDS::DynamicDataInfo::stored_size `[get]`

The number of data members in this **DDS::DynamicData** (p. 719) sample.

6.56 DDS::DynamicDataMemberInfo Class Reference

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

`#include <managed_dynamicdata.h>`

Properties

^ System::Int32 **member_id** [get]

*An integer that uniquely identifies the data member within this **DDS::DynamicData** (p. 719) sample's type.*

^ System::String^ **member_name** [get]

The string name of the data member.

^ System::Boolean **member_exists** [get]

Indicates whether the corresponding member of the data type actually exists in this sample.

^ **TCKind member_kind** [get]

The kind of type of this data member (e.g. integer, structure, etc.).

^ System::UInt32 **element_count** [get]

The number of elements within this data member.

^ **TCKind element_kind** [get]

The kind of type of the elements within this data member.

6.56.1 Detailed Description

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

See also:

DDS::DynamicData::get_member_info (p. 745)

6.56.2 Property Documentation

6.56.2.1 System:: Int32 DDS::DynamicDataMemberInfo::member_id [get]

An integer that uniquely identifies the data member within this **DDS::DynamicData** (p. 719) sample's type.

For sparse data types, this value will be assigned by the type designer. For types defined in IDL, it will be assigned automatically by the middleware based on the member's declaration order within the type.

See also:

DDS::TCKind

6.56.2.2 System:: String^ DDS::DynamicDataMemberInfo::member_name [get]

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.

See also:

DDS::DynamicDataMemberInfo::representation_count

6.56.2.3 System:: Boolean DDS::DynamicDataMemberInfo::member_exists [get]

Indicates whether the corresponding member of the data type actually exists in this sample.

For non-sparse data types, this value will always be true.

See also:

DDS::TCKind

6.56.2.4 TCKind DDS::DynamicDataMemberInfo::member_kind [get]

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. 1301) and getting the DDS::TCKind from there.

6.56.2.5 System:: UInt32 DDS::DynamicDataMemberInfo::element_count [get]

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.

6.56.2.6 TCKind DDS::DynamicDataMemberInfo::element_kind [get]

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::TCKind::TK_NULL here.

6.57 DDS::DynamicDataProperty_t Class Reference

A collection of attributes used to configure **DDS::DynamicData** (p. 719) objects.

`#include <managed_dynamicdata.h>`

Public Attributes

- System::Int32 **buffer_initial_size**

  *The initial amount of memory used by this **DDS::DynamicData** (p. 719) object, in bytes.*

- System::Int32 **buffer_max_size**

  *The maximum amount of memory that this **DDS::DynamicData** (p. 719) object may use, in bytes.*

6.57.1 Detailed Description

A collection of attributes used to configure **DDS::DynamicData** (p. 719) objects.

6.57.2 Member Data Documentation

6.57.2.1 System::Int32 DDS::DynamicDataProperty_t::buffer_initial_size

The initial amount of memory used by this **DDS::DynamicData** (p. 719) object, in bytes.

See also:

DDS::DynamicDataProperty_t::buffer_max_size (p. 813)

6.57.2.2 System::Int32 DDS::DynamicDataProperty_t::buffer_max_size

The maximum amount of memory that this **DDS::DynamicData** (p. 719) object may use, in bytes.

It will grow to this size from the initial size as needed.

See also:

DDS::DynamicDataProperty_t::buffer_initial_size (p. 813)

6.58 DDS::DynamicDataReader Class Reference

Reads (subscribes to) objects of type **DDS::DynamicData** (p. 719).

`#include <managed_dynamicdata.h>`

Inheritance diagram for DDS::DynamicDataReader::

6.58.1 Detailed Description

Reads (subscribes to) objects of type **DDS::DynamicData** (p. 719).

Instantiates `DDS::DataReader` (p. 433) < **DDS::DynamicData** (p. 719) > .

See also:

DDS::DataReader (p. 433)
DDS::TypedDataReader (p. 1338)
DDS::DynamicData (p. 719)

6.59 DDS::DynamicDataSeq Class Reference

An ordered collection of **DDS::DynamicData** (p. 719) elements.

`#include <managed_dynamicdata.h>`

Public Member Functions

- ^ **DynamicDataSeq** ()

  *Construct a new empty **DDS::DynamicDataSeq** (p. 816).*

- ^ **DynamicDataSeq** (System::Int32 max)

  *Construct a new empty **DDS::DynamicDataSeq** (p. 816).*

- ^ **DynamicDataSeq** (**Sequence**< **DynamicData**^ >^src)

  *Construct a new **DDS::DynamicDataSeq** (p. 816) containing the same elements as the given collection.*

6.59.1 Detailed Description

An ordered collection of **DDS::DynamicData** (p. 719) elements.

Instantiates `DDS::Sequence` (p. 1163) < **DDS::DynamicData** (p. 719) > .

See also:

DDS::Sequence (p. 1163)
DDS::DynamicData (p. 719)

6.59.2 Constructor & Destructor Documentation

6.59.2.1 DDS::DynamicDataSeq::DynamicDataSeq () [inline]

Construct a new empty **DDS::DynamicDataSeq** (p. 816).

6.59.2.2 DDS::DynamicDataSeq::DynamicDataSeq (System::Int32 *max*) [inline]

Construct a new empty **DDS::DynamicDataSeq** (p. 816).

The new sequence will have the given maximum.

See also:

DDS::Sequence::maximum (p. 1172)

6.59.2.3 DDS::DynamicDataSeq::DynamicDataSeq (Sequence< DynamicData^ >^ *src*) [inline]

Construct a new **DDS::DynamicDataSeq** (p. 816) containing the same elements as the given collection.

6.60 DDS::DynamicDataTypeProperty_t Class Reference

A collection of attributes used to configure **DDS::DynamicDataTypeSupport** (p. 822) objects.

`#include <managed_dynamicdata.h>`

Public Attributes

- **DynamicDataProperty_t^ data**

  *These properties will be provided to every new **DDS::DynamicData** (p. 719) sample created from the **DDS::DynamicDataTypeSupport** (p. 822).*

- **DynamicDataTypeSerializationProperty_t^ serialization**

  Properties that govern how the data of this type will be serialized on the network.

Static Public Attributes

- static **DynamicDataTypeProperty_t^ DYNAMIC_DATA_TYPE_PROPERTY_DEFAULT**

  *Sentinel constant indicating default values for **DDS::DynamicDataTypeProperty_t** (p. 818).*

6.60.1 Detailed Description

A collection of attributes used to configure **DDS::DynamicDataTypeSupport** (p. 822) objects.

The properties of a **DDS::DynamicDataTypeSupport** (p. 822) object contain the properties that will be used to instantiate any samples created by that object.

6.60.2 Member Data Documentation

6.60.2.1 DynamicDataProperty_t ^ DDS::DynamicDataTypeProperty_t::data

These properties will be provided to every new **DDS::DynamicData** (p. 719) sample created from the **DDS::DynamicDataTypeSupport** (p. 822).

6.60.2.2 DynamicDataTypeSerializationProperty_t ^ DDS::DynamicDataTypeProperty_t::serialization

Properties that govern how the data of this type will be serialized on the network.

6.61 DDS::DynamicDataTypeSerializationProperty_t Class Reference

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

`#include <managed_dynamicdata.h>`

Public Attributes

- System::Boolean **use_42e_compatible_alignment**

  Use RTI Data Distribution Service 4.2e-compatible alignment for large primitive types.

- System::UInt32 **max_size_serialized**

  The maximum number of bytes that objects of a given type could consume when serialized on the network.

6.61.1 Detailed Description

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

6.61.2 Member Data Documentation

6.61.2.1 System::Boolean DDS::DynamicDataTypeSerializationProperty_t::use_42e_compatible_alignment

Use RTI Data Distribution Service 4.2e-compatible alignment for large primitive types.

In RTI Data Distribution Service 4.2e, the default alignment for large primitive types – System::Int64, System::UInt64, System::Double, and **DDS::LongDouble** (p. 976) – was not RTPS-compliant. This compatibility mode allows applications targeting post-4.2e versions of RTI Data Distribution Service 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 Data Distribution Service 4.2e.

6.61.2.2 System::UInt32 DDS::DynamicDataTypeSerializationProperty_t::max_size_serialized

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. 1301), whichever is smaller.

6.62 DDS::DynamicDataTypeSupport Class Reference

A factory for registering a dynamically defined type and creating **DDS::DynamicData** (p. 719) objects.

`#include <managed_dynamicdata.h>`

Inheritance diagram for DDS::DynamicDataTypeSupport::

Public Member Functions

- ^ **DynamicDataTypeSupport** (**TypeCode**^ type, **DynamicDataTypeProperty_t**^ props)

  *Construct a new **DDS::DynamicDataTypeSupport** (p. 822) object.*

- ^ virtual ~**DynamicDataTypeSupport** ()

  *Delete a **DDS::DynamicDataTypeSupport** (p. 822) object.*

- ^ System::Boolean **is_valid** ()

  Indicates whether the object was constructed properly.

- ^ void **register_type** (**DomainParticipant**^ participant, System::String^ type_name)

  *Associate the **DDS::TypeCode** (p. 1301) with the given **DDS::DomainParticipant** (p. 577) under the given logical name.*

- ^ void **unregister_type** (**DomainParticipant**^ participant, System::String^ type_name)

  *Remove the definition of this type from the **DDS::DomainParticipant** (p. 577).*

- ^ System::String^ **get_type_name** ()

  Get the default name of this type.

- ^ **TypeCode**^ **get_data_type** ()

  *Get the **DDS::TypeCode** (p. 1301) wrapped by this **DDS::DynamicDataTypeSupport** (p. 822).*

- ^ **DynamicData**^ **create_data** ()

  *Create a new **DDS::DynamicData** (p. 719) sample initialized with the **DDS::TypeCode** (p. 1301) and properties of this **DDS::DynamicDataTypeSupport** (p. 822).*

^ void **delete_data** (**DynamicData**^ a_data)

*Finalize and deallocate the **DDS::DynamicData** (p. 719) sample.*

^ void **copy_data** (**DynamicData**^ dest, **DynamicData**^ source)

Deeply copy the given data samples.

6.62.1 Detailed Description

A factory for registering a dynamically defined type and creating **DDS::DynamicData** (p. 719) objects.

A **DDS::DynamicDataTypeSupport** (p. 822) has three roles:

1. It associates a **DDS::TypeCode** (p. 1301) with policies for managing objects of that type. See the constructor, **DDS::DynamicDataTypeSupport::DynamicDataTypeSupport** (p. 823).
2. It registers its type under logical names with a **DDS::DomainParticipant** (p. 577). See **DDS::DynamicDataTypeSupport::register_type** (p. 825).
3. It creates **DDS::DynamicData** (p. 719) samples pre-initialized with the type and properties of the type support itself. See **DDS::DynamicDataTypeSupport::create_data** (p. 826).

6.62.2 Constructor & Destructor Documentation

6.62.2.1 DDS::DynamicDataTypeSupport::DynamicDataTypeSupport (TypeCode^ *type*, DynamicDataTypeProperty_t^ *props*)

Construct a new **DDS::DynamicDataTypeSupport** (p. 822) object.

This step is usually followed by type registration.

Parameters:

type The **DDS::TypeCode** (p. 1301) that describes the members of this type.

props Policies that describe how to manage the memory and other properties of the data samples created by this factory. In most cases, the default values will be appropriate; see **DDS::DynamicDataTypeProperty_t.DYNAMIC_DATA_TYPE_PROPERTY_DEFAULT** (p. 76).

See also:

DDS::DynamicDataTypeSupport::register_type (p. 825)

6.62.2.2 virtual DDS::DynamicDataTypeSupport::~DynamicDataTypeSupport () [inline, virtual]

Delete a **DDS::DynamicDataTypeSupport** (p. 822) object.

A **DDS::DynamicDataTypeSupport** (p. 822) cannot be deleted while it is still in use. For each **DDS::DomainParticipant** (p. 577) with which the **DDS::DynamicDataTypeSupport** (p. 822) is registered, either the type must be unregistered or the participant must be deleted.

See also:

DDS::DynamicDataTypeSupport::unregister_type (p. 825)
DDS::DynamicDataTypeSupport::DynamicDataTypeSupport (p. 823)

6.62.3 Member Function Documentation

6.62.3.1 System::Boolean DDS::DynamicDataTypeSupport::is_valid ()

Indicates whether the object was constructed properly.

This method returns true if the constructor succeeded; it returns false if the constructor failed for any reason, which should also have resulted in a log message.

Possible failure reasons include passing an invalid type or invalid properties to the constructor.

This method is necessary because C++ exception support is not consistent across all of the platforms on which RTI Data Distribution Service runs. Therefore, the implementation does not throw any exceptions in the constructor.

See also:

DDS::DynamicDataTypeSupport::DynamicDataTypeSupport (p. 823)

6.62.3.2 void DDS::DynamicDataTypeSupport::register_type (DomainParticipant^ *participant*, System::String^ *type_name*)

Associate the **DDS::TypeCode** (p. 1301) with the given **DDS::DomainParticipant** (p. 577) 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.

See also:

FooTypeSupport::register_type (p. 885)
DDS::DomainParticipant::create_topic (p. 621)
DDS::DynamicDataTypeSupport::unregister_type (p. 825)

6.62.3.3 void DDS::DynamicDataTypeSupport::unregister_type (DomainParticipant^ *participant*, System::String^ *type_name*)

Remove the definition of this type from the **DDS::DomainParticipant** (p. 577).

This operation is optional; all types are automatically unregistered when a **DDS::DomainParticipant** (p. 577) 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. 1258) is still referring to it.

See also:

FooTypeSupport::unregister_type (p. 886)
DDS::DynamicDataTypeSupport::register_type (p. 825)

6.62.3.4 System::String ^ DDS::DynamicDataTypeSupport::get_type_name ()

Get the default name of this type.

The **DDS::TypeCode** (p. 1301) that is wrapped by this **DDS::DynamicDataTypeSupport** (p. 822) 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.

In C#:

```
myTypeSupport.register_type(myParticipant, myTypeSupport.get_type_name());
```

In C++/CLI:

```
myTypeSupport->register_type(myParticipant, myTypeSupport->get_type_name());
```

See also:

FooTypeSupport::get_type_name (p. 885)

6.62.3.5 TypeCode ^ DDS::DynamicDataTypeSupport::get_data_type ()

Get the **DDS::TypeCode** (p. 1301) wrapped by this **DDS::DynamicDataTypeSupport** (p. 822).

6.62.3.6 DynamicData ^ DDS::DynamicDataTypeSupport::create_data ()

Create a new **DDS::DynamicData** (p. 719) sample initialized with the **DDS::TypeCode** (p. 1301) and properties of this **DDS::DynamicDataTypeSupport** (p. 822).

You must delete your **DDS::DynamicData** (p. 719) object when you are finished with it.

In C#:

```
DynamicData sample = myTypeSupport.create_data();
 // Do something...
 myTypeSupport.delete_data(sample);
```

In C++/CLI:

```
DynamicData^ sample = myTypeSupport->create_data();
 // Do something...
 myTypeSupport->delete_data(sample);
```

See also:

DDS::DynamicDataTypeSupport::delete_data (p. 827)
FooTypeSupport::create_data (p. 887)
DDS::DynamicData::DynamicData (p. 734)
DDS::DynamicDataTypeProperty_t::data (p. 819)

6.62.3.7 void DDS::DynamicDataTypeSupport::delete_data (DynamicData^ *a_data*)

Finalize and deallocate the **DDS::DynamicData** (p. 719) sample.

See also:

FooTypeSupport::delete_data (p. 888)
DDS::DynamicDataTypeSupport::create_data (p. 826)

6.62.3.8 void DDS::DynamicDataTypeSupport::copy_data (DynamicData^ *dest*, DynamicData^ *source*)

Deeply copy the given data samples.

6.63 DDS::DynamicDataWriter Class Reference

Writes (publishes) objects of type **DDS::DynamicData** (p. 719).

`#include <managed_dynamicdata.h>`

Inheritance diagram for DDS::DynamicDataWriter::

Public Member Functions

- ^ **InstanceHandle_t register_instance** (**DynamicData**^ instance_data)

  Informs RTI Data Distribution Service that the application will be modifying a particular instance.

- ^ **InstanceHandle_t register_instance_w_timestamp** (**DynamicData**^ instance_data, **DDS::Time_t**% source_timestamp)

  Performs the same functions as register_instance except that the application provides the value for the `source_timestamp`.

- ^ void **unregister_instance** (**DynamicData**^ instance_data, **DDS::InstanceHandle_t**% handle)

  Reverses the action of ***DDS::TypedDataWriter::register_instance*** *(p. 1370).*

- ^ void **unregister_instance_w_timestamp** (**DynamicData**^ instance_data, **DDS::InstanceHandle_t**% handle, **DDS::Time_t**% source_timestamp)

  Performs the same function as ***DDS::TypedDataWriter::unregister_instance*** *(p. 1372) except that it also provides the value for the* `source_timestamp`.

- ^ void **write** (**DynamicData**^ instance_data, **DDS::InstanceHandle_t**% handle)

  Modifies the value of a data instance.

- ^ void **write_w_timestamp** (**DynamicData**^ instance_data, **DDS::InstanceHandle_t**% handle, **DDS::Time_t**% source_timestamp)

  Performs the same function as ***DDS::TypedDataWriter::write*** *(p. 1376) except that it also provides the value for the* `source_timestamp`.

^ void **dispose** (**DynamicData**^ instance_data, **DDS::InstanceHandle_t**% instance_handle)

Requests the middleware to delete the data.

^ void **dispose_w_timestamp** (**DynamicData**^ instance_data, **DDS::InstanceHandle_t**% instance_handle, **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. 433) objects by means of the* `source_timestamp` *attribute inside the* ***DDS::SampleInfo*** *(p. 1148).*

^ void **get_key_value** (**DynamicData**^ key_holder, **DDS::InstanceHandle_t**% handle)

Retrieve the instance `key` *that corresponds to an instance* `handle`*.*

^ **InstanceHandle_t lookup_instance** (**DynamicData**^ key_holder)

Retrieve the instance `handle` *that corresponds to an instance* `key_holder`*.*

6.63.1 Detailed Description

Writes (publishes) objects of type **DDS::DynamicData** (p. 719).

Instantiates `DDS::DataWriter` (p. 499) < **DDS::DynamicData** (p. 719) > .

See also:

DDS::DataWriter (p. 499)
DDS::TypedDataWriter (p. 1368)
DDS::DynamicData (p. 719)

6.63.2 Member Function Documentation

6.63.2.1 InstanceHandle_t DDS::DynamicDataWriter::register_instance (DynamicData^ *instance_data*)

Informs RTI Data Distribution Service 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::InstanceHandle_t::HANDLE_NIL** (p. 53). The operation takes as a parameter an instance (of which only the key

value is examined) and returns a `handle` that can be used in successive **write()** (p. 835) or **dispose()** (p. 839) operations.

The operation gives RTI Data Distribution Service 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::InstanceHandle_t::HANDLE_NIL** (p. 53) as the **DDS::InstanceHandle_t** (p. 905) paramater to the write or dispose operation and RTI Data Distribution Service will auto-register the instance.

For best performance, the operation should be invoked prior to calling any operation that modifies the instance, such as **DDS::TypedDataWriter::write** (p. 1376), **DDS::TypedDataWriter::write_w_timestamp** (p. 1378), **DDS::TypedDataWriter::dispose** (p. 1379) and **DDS::TypedDataWriter::dispose_w_timestamp** (p. 1381) and the handle used in conjunction with the data for those calls.

When this operation is used, RTI Data Distribution Service will automatically supply the value of the `source_timestamp` that is used.

This operation may fail and return **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) if **DDS::ResourceLimitsQosPolicy::max_instances** (p. 1112) 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. 499) has been enabled. Otherwise, **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) will be returned.

Parameters:

instance_data *<<in>> (p. 175)* 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. 905), such as write, dispose, unregister_instance, or return **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) on failure. If the instance_data is of a data type that has no keys, this function always return **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53).

See also:

DDS::TypedDataWriter::unregister_instance (p. 1372),

DDS::TypedDataWriter::get_key_value (p. 1383), **RELATIONSHIP BETWEEN REGISTRATION, LIVELINESS and OWNERSHIP** (p. 995)

6.63.2.2 InstanceHandle_t DDS::DynamicDataWriter::register_instance_w_timestamp (DynamicData^ *instance_data*, 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. 292) QoS policy for details.

This operation may fail and return **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) if **DDS::ResourceLimitsQosPolicy::max_instances** (p. 1112) limit has been exceeded.

This operation can only be called after **DDS::DataWriter** (p. 499) has been enabled. Otherwise, **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) will be returned.

Parameters:

instance_data *<<in>> (p. 175)* The instance that should be registered. Of this instance, only the fields that represent the key are examined by the function. Cannot be NULL.

source_timestamp *<<in>> (p. 175)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation (used in a *register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application provided timestamp). This timestamp may potentially affect the order in which readers observe events from multiple writers. 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. 905), such as write, dispose, unregister_instance, or return **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) on failure. If the instance_data is of a data type that has no keys, this function always return **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53).

See also:

DDS::TypedDataWriter::unregister_instance (p. 1372), **DDS::TypedDataWriter::get_key_value** (p. 1383)

6.63.2.3 void DDS::DynamicDataWriter::unregister_instance (DynamicData^ *instance_data*, DDS::InstanceHandle_t% *handle*)

Reverses the action of **DDS::TypedDataWriter::register_instance** (p. 1370).

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 **DDS::TypedDataWriter::register_instance** (p. 1370). Otherwise, this operation may fail with **DDS::Retcode_BadParameter** (p. 1115).

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 Data Distribution Service will automatically supply the value of the `source_timestamp` that is used.

This operation informs RTI Data Distribution Service that the **DDS::DataWriter** (p. 499) is no longer going to provide any information about the instance. This operation also indicates that RTI Data Distribution Service can locally remove all information regarding that instance. The application should not attempt to use the `handle` previously allocated to that instance after calling **DDS::TypedDataWriter::unregister_instance()** (p. 1372).

The special value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) 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::InstanceHandle_t::HANDLE_NIL** (p. 53), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with **DDS::Retcode_BadParameter** (p. 1115).

RTI Data Distribution Service will not detect the error when the `handle` is any value other than **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), 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 Data Distribution Service will treat as if the **unregister_instance()** (p. 832) operation is for the instance as indicated by the `handle`.

If after a **DDS::TypedDataWriter::unregister_instance** (p. 1372), the application wants to modify (**DDS::TypedDataWriter::write** (p. 1376) or

DDS::TypedDataWriter::dispose (p. 1379)) an instance, it has to register it again, or else use the special `handle` value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53).

This operation does not indicate that the instance is deleted (that is the purpose of **DDS::TypedDataWriter::dispose** (p. 1379)). The operation **DDS::TypedDataWriter::unregister_instance** (p. 1372) just indicates that the **DDS::DataWriter** (p. 499) no longer has anything to say about the instance. **DDS::DataReader** (p. 433) entities that are reading the instance may receive a sample with **DDS::InstanceStateKind::NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 909) for the instance, unless there are other **DDS::DataWriter** (p. 499) objects writing that same instance.

This operation can affect the ownership of the data instance (see **OWNERSHIP** (p. 283)). If the **DDS::DataWriter** (p. 499) was the exclusive owner of the instance, then calling **unregister_instance()** (p. 832) will relinquish that ownership.

If **DDS::ReliabilityQosPolicy::kind** (p. 1097) is set to DDS::ReliabilityQosPolicyKind::RELIABLE_RELIABILITY_QOS 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. 1097); if this writer is still unable to unregister after that period, this method will fail with **DDS::Retcode_Timeout** (p. 1124).

Parameters:

instance_data *<<in>>* *(p. 175)* The instance that should be unregistered. If **Foo** (p. 877) *has* a key and `instance_handle` is **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used. If `instance_data` is used, it must represent an instance that has been registerd. Otherwise, this method may fail with **DDS::Retcode_BadParameter** (p. 1115) . If **Foo** (p. 877) *has* a key, `instance_data` can be NULL only if `handle` is not **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53). Otherwise, this method will fail with **DDS::Retcode_BadParameter** (p. 1115).

handle *<<in>>* *(p. 175)* represents the `instance` to be unregistered. If **Foo** (p. 877) *has* a key and `handle` is **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), `handle` is not used and `instance` is deduced from `instance_data`. If **Foo** (p. 877) has no key, `handle` is not used. If `handle` is used, it must represent an instance that has been registered. Otherwise, this method may fail with **DDS::Retcode_BadParameter** (p. 1115). This method will fail with **DDS::Retcode_BadParameter** (p. 1115) if `handle` is NULL. If **Foo** (p. 877) *has* a key, `handle` cannot be **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) if `instance_data` is NULL. Otherwise, this method will report the error **DDS::Retcode_BadParameter**

(p. 1115).

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_Timeout** (p. 1124) or **DDS::Retcode_NotEnabled** (p. 1121)

See also:

DDS::TypedDataWriter::register_instance (p. 1370)
DDS::TypedDataWriter::unregister_instance_w_timestamp (p. 1374)
DDS::TypedDataWriter::get_key_value (p. 1383)
RELATIONSHIP BETWEEN REGISTRATION, LIVELINESS and OWNERSHIP (p. 995)

6.63.2.4 void DDS::DynamicDataWriter::unregister_instance_w_timestamp (DynamicData^ *instance_data*, DDS::InstanceHandle_t% *handle*, DDS::Time_t% *source_timestamp*)

Performs the same function as **DDS::TypedDataWriter::unregister_instance** (p. 1372) 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. 292) QoS policy for details.

The constraints on the values of the `handle` parameter and the corresponding error behavior are the same specified for the **DDS::TypedDataWriter::unregister_instance** (p. 1372) operation.

This operation may block and may time out (**DDS::Retcode_Timeout** (p. 1124)) under the same circumtances described for the unregister_instance operation.

Parameters:

instance_data *<<in>>* *(p. 175)* The instance that should be unregistered. If **Foo** (p. 877) *has* a key and `instance_handle` is **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used. If `instance_data` is used, it must represent an instance that has been registerd. Otherwise, this method may fail with **DDS::Retcode_BadParameter** (p. 1115). If **Foo** (p. 877) *has* a key, `instance_data` can be NULL only if `handle` is

not **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53). Otherwise, this method will fail with **DDS::Retcode_BadParameter** (p. 1115).

handle *<<in>>* *(p. 175)* represents the `instance` to be unregistered. If **Foo** (p. 877) *has* a key and `handle` is **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), `handle` is not used and `instance` is deduced from `instance_data`. If **Foo** (p. 877) has no key, `handle` is not used. If `handle` is used, it must represent an instance that has been registered. Otherwise, this method may fail with **DDS::Retcode_BadParameter** (p. 1115). This method will fail with **DDS::Retcode_BadParameter** (p. 1115) if `handle` is NULL. If **Foo** (p. 877) *has* a key, `handle` cannot be **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) if `instance_data` is NULL. Otherwise, this method will fail with **DDS::Retcode_BadParameter** (p. 1115).

source_timestamp *<<in>>* *(p. 175)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation (used in a *register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application provided timestamp). This timestamp may potentially affect the order in which readers observe events from multiple writers. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_Timeout** (p. 1124) or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::TypedDataWriter::register_instance (p. 1370)
DDS::TypedDataWriter::unregister_instance (p. 1372)
DDS::TypedDataWriter::get_key_value (p. 1383)

6.63.2.5 void DDS::DynamicDataWriter::write (DynamicData^ *instance_data*, DDS::InstanceHandle_t% *handle*)

Modifies the value of a data instance.

When this operation is used, RTI Data Distribution Service will automatically supply the value of the `source_timestamp` that is made available to **DDS::DataReader** (p. 433) objects by means of the `source_timestamp` attribute inside the **DDS::SampleInfo** (p. 1148). (Refer to **DDS::SampleInfo** (p. 1148) and **DESTINATION_ORDER** (p. 292) QoS policy for details).

As a side effect, this operation asserts liveliness on the **DDS::DataWriter** (p. 499) itself, the **DDS::Publisher** (p. 1044) and the **DDS::DomainParticipant** (p. 577).

Note that the special value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) 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::InstanceHandle_t::HANDLE_NIL** (p. 53), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with **DDS::Retcode_BadParameter** (p. 1115).

RTI Data Distribution Service will not detect the error when the `handle` is any value other than **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), 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 Data Distribution Service will treat as if the **write()** (p. 835) operation is for the instance as indicated by the `handle`.

This operation may block if the **RELIABILITY** (p. 290) `kind` is set to DDS::ReliabilityQosPolicyKind::RELIABLE_RELIABILITY_QOS and the modification would cause data to be lost or else cause one of the limits specified in the **RESOURCE_LIMITS** (p. 298) to be exceeded.

Specifically, this operation may block in the following situations (note that the list may not be exhaustive), even if its DDS::HistoryQosPolicyKind is DDS::HistoryQosPolicyKind::KEEP_LAST_HISTORY_QOS:

- If (**DDS::ResourceLimitsQosPolicy::max_samples** (p. 1112) < **DDS::ResourceLimitsQosPolicy::max_instances** (p. 1112) * **DDS::HistoryQosPolicy::depth** (p. 901)), then in the situation where the `max_samples` resource limit is exhausted, RTI Data Distribution Service is allowed to discard samples of some other instance, as long as at least one sample remains for such an instance. If it is still not possible to make space available to store the modification, the writer is allowed to block.
- If (**DDS::ResourceLimitsQosPolicy::max_samples** (p. 1112) < **DDS::ResourceLimitsQosPolicy::max_instances** (p. 1112)), then the **DataWriter** (p. 499) may block regardless of the **DDS::HistoryQosPolicy::depth** (p. 901).

This operation may also block when using DDS::ReliabilityQosPolicyKind::BEST_EFFORT_RELIABILITY_QOS and DDS::PublishModeQosPolicyKind::ASYNCHRONOUS_PUBLISH_MODE_QOS. In this case, the **DDS::DataWriter** (p. 499) will queue samples until they are sent by the asynchronous publishing thread. The number of samples that can be stored is determined by the **DDS::HistoryQosPolicy** (p. 898). If the asynchronous thread does not send samples fast enough (e.g., when using a slow **DDS::FlowController** (p. 867)), the queue may fill up. In that case,

subsequent write calls will block.

If this operation *does* block for any of the above reasons, the **RELIABILITY** (p. 290) `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** (p. 499) is able to store the modification without exceeding the limits, the operation will time out (**DDS::Retcode_Timeout** (p. 1124)).

If there are no instance resources left, this operation may fail with **DDS::Retcode_OutOfResources** (p. 1122). Calling **DDS::TypedDataWriter::unregister_instance** (p. 1372) may help freeing up some resources.

This operation will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123) 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).

Parameters:

instance_data *<<in>> (p. 175)* The data to write.

This method will fail with **DDS::Retcode_BadParameter** (p. 1115) if `instance_data` is NULL.

Parameters:

handle *<<in>> (p. 175)* Either the handle returned by a previous call to **DDS::TypedDataWriter::register_instance** (p. 1370), or else the special value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53). If **Foo** (p. 877) *has* a key and `handle` is not **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), `handle` must represent a registered instance of type **Foo** (p. 877). Otherwise, this method may fail with **DDS::Retcode_BadParameter** (p. 1115). This method will fail with **DDS::Retcode_BadParameter** (p. 1115) if `handle` is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_Timeout** (p. 1124), **DDS::Retcode_PreconditionNotMet** (p. 1123), **DDS::Retcode_OutOfResources** (p. 1122), or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::DataReader (p. 433)
DDS::TypedDataWriter::write_w_timestamp (p. 1378)
DESTINATION_ORDER (p. 292)

6.63.2.6 void DDS::DynamicDataWriter::write_w_timestamp (DynamicData^ *instance_data*, DDS::InstanceHandle_t% *handle*, DDS::Time_t% *source_timestamp*)

Performs the same function as **DDS::TypedDataWriter::write** (p. 1376) except that it also provides the value for the `source_timestamp`.

Explicitly provides the timestamp that will be available to the **DDS::DataReader** (p. 433) objects by means of the `source_timestamp` attribute inside the **DDS::SampleInfo** (p. 1148). (Refer to **DDS::SampleInfo** (p. 1148) and **DESTINATION_ORDER** (p. 292) QoS policy for details)

The constraints on the values of the `handle` parameter and the corresponding error behavior are the same specified for the **DDS::TypedDataWriter::write** (p. 1376) operation.

This operation may block and time out (**DDS::Retcode_Timeout** (p. 1124)) under the same circumtances described for **DDS::TypedDataWriter::write** (p. 1376).

If there are no instance resources left, this operation may fail with **DDS::Retcode_OutOfResources** (p. 1122). Calling **DDS::TypedDataWriter::unregister_instance** (p. 1372) may help free up some resources.

This operation may fail with **DDS::Retcode_BadParameter** (p. 1115) under the same circumstances described for the write operation.

Parameters:

instance_data *<<in>> (p. 175)* The data to write. This method will fail with **DDS::Retcode_BadParameter** (p. 1115) if `instance_data` is NULL.

handle *<<in>> (p. 175)* Either the handle returned by a previous call to **DDS::TypedDataWriter::register_instance** (p. 1370), or else the special value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53). If **Foo** (p. 877) *has* a key and `handle` is not **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), `handle` must represent a registered instance of type **Foo** (p. 877). Otherwise, this method may fail with **DDS::Retcode_BadParameter** (p. 1115). This method will fail with **DDS::Retcode_BadParameter** (p. 1115) if `handle` is NULL.

source_timestamp *<<in>> (p. 175)* When using DDS::DestinationOrderQosPolicyKind::BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS the timestamp value must be greater than or equal to the timestamp value used in the last writer operation (*register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application-provided timestamp) However, if it is less than the

timestamp of the previous operation but the difference is less than the **DDS::DestinationOrderQosPolicy::source_timestamp_tolerance** (p. 562), 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. 562), the function will return **DDS::Retcode_BadParameter** (p. 1115).

Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_Timeout** (p. 1124), **DDS::Retcode_OutOfResources** (p. 1122), or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::TypedDataWriter::write (p. 1376)
DDS::DataReader (p. 433)
DESTINATION_ORDER (p. 292)

6.63.2.7 void DDS::DynamicDataWriter::dispose (DynamicData^ *instance_data*, DDS::InstanceHandle_t% *instance_handle*)

Requests the middleware to delete the data.

This operation is useful only for keyed data types. Using it for non-keyed types has no effect and reports no error.

The actual deletion is postponed until there is no more use for that data in the whole system.

Applications are made aware of the deletion by means of operations on the **DDS::DataReader** (p. 433) objects that already knew that instance. **DDS::DataReader** (p. 433) objects that didn't know the instance will never see it.

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 Data Distribution Service will automatically supply the value of the `source_timestamp` that is made available to **DDS::DataReader** (p. 433) objects by means of the `source_timestamp` attribute inside the **DDS::SampleInfo** (p. 1148).

The constraints on the values of the handle parameter and the corresponding error behavior are the same specified for the **DDS::TypedDataWriter::unregister_instance** (p. 1372) operation.

The special value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) 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 `handle` is any value other than **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with **DDS::Retcode_BadParameter** (p. 1115).

RTI Data Distribution Service will not detect the error when the `handle` is any value other than **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), 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 Data Distribution Service will treat as if the **dispose()** (p. 839) operation is for the instance as indicated by the `handle`.

This operation may block and time out (**DDS::Retcode_Timeout** (p. 1124)) under the same circumtances described for **DDS::TypedDataWriter::write()** (p. 1376).

If there are no instance resources left, this operation may fail with **DDS::Retcode_OutOfResources** (p. 1122). Calling **DDS::TypedDataWriter::unregister_instance** (p. 1372) may help freeing up some resources.

Parameters:

instance_data *<<in>>* *(p. 175)* The data to dispose. If **Foo** (p. 877) *has* a key and `instance_handle` is **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used. If **Foo** (p. 877) *has* a key, `instance_data` can be NULL only if `instance_handle` is not **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53). Otherwise, this method will fail with **DDS::Retcode_BadParameter** (p. 1115).

instance_handle *<<in>>* *(p. 175)* Either the handle returned by a previous call to **DDS::TypedDataWriter::register_instance** (p. 1370), or else the special value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53). If **Foo** (p. 877) *has* a key and `instance_handle` is **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), `instance_handle` is not used and `instance` is deduced from `instance_data`. If **Foo** (p. 877) has no key, `instance_handle` is not used. If `handle` is used, it must represent a registered instance of type **Foo** (p. 877). Otherwise, this method fail with **DDS::Retcode_BadParameter** (p. 1115). This method will fail with **DDS::Retcode_BadParameter** (p. 1115) if `handle` is NULL. If **Foo** (p. 877) *has* a key, `instance_handle` can-

not be **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) if `instance_data` is NULL. Otherwise, this method will fail with **DDS::Retcode_BadParameter** (p. 1115).

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_Timeout** (p. 1124), **DDS::Retcode_OutOfResources** (p. 1122) or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::TypedDataWriter::dispose_w_timestamp (p. 1381)
RELATIONSHIP BETWEEN REGISTRATION, LIVELINESS and OWNERSHIP (p. 995)

6.63.2.8 void DDS::DynamicDataWriter::dispose_w_timestamp (DynamicData^ *instance_data*, DDS::InstanceHandle_t% *instance_handle*, 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. 433) objects by means of the `source_timestamp` attribute inside the **DDS::SampleInfo** (p. 1148).

The constraints on the values of the `handle` parameter and the corresponding error behavior are the same specified for the **DDS::TypedDataWriter::dispose** (p. 1379) operation.

This operation may block and time out (**DDS::Retcode_Timeout** (p. 1124)) under the same circumtances described for **DDS::TypedDataWriter::write** (p. 1376).

If there are no instance resources left, this operation may fail with **DDS::Retcode_OutOfResources** (p. 1122). Calling **DDS::TypedDataWriter::unregister_instance** (p. 1372) may help freeing up some resources.

Parameters:

instance_data *<<in>> (p. 175)* The data to dispose. If **Foo** (p. 877) *has* a key and `instance_handle` is **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used. If **Foo** (p. 877) *has* a key, `instance_data` can be NULL only if `instance_handle` is not **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53). Otherwise, this method will fail with **DDS::Retcode_BadParameter** (p. 1115).

instance_handle *<<in>>* *(p. 175)* Either the handle returned by a previous call to **DDS::TypedDataWriter::register_instance** (p. 1370), or else the special value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53). If **Foo** (p. 877) *has* a key and `instance_handle` is **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), `instance_handle` is not used and `instance` is deduced from `instance_data`. If **Foo** (p. 877) has no key, `instance_handle` is not used. If `handle` is used, it must represent a registered instance of type **Foo** (p. 877). Otherwise, this method may fail with **DDS::Retcode_BadParameter** (p. 1115) This method will fail with **DDS::Retcode_BadParameter** (p. 1115) if `handle` is NULL. If **Foo** (p. 877) *has* a key, `instance_handle` cannot be **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) if `instance_data` is NULL. Otherwise, this method will fail with **DDS::Retcode_BadParameter** (p. 1115).

source_timestamp *<<in>>* *(p. 175)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation (used in a *register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application provided timestamp). This timestamp may potentially affect the order in which readers observe events from multiple writers. This timestamp will be available to the **DDS::DataReader** (p. 433) objects by means of the `source_timestamp` attribute inside the **DDS::SampleInfo** (p. 1148). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_Timeout** (p. 1124), **DDS::Retcode_OutOfResources** (p. 1122) or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::TypedDataWriter::dispose (p. 1379)

6.63.2.9 void DDS::DynamicDataWriter::get_key_value (DynamicData^ *key_holder*, 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 **Foo** (p. 877) has no key, this method has no effect and exit with no error.

For keyed data types, this operation may fail with **DDS::Retcode_-BadParameter** (p. 1115) if the `handle` does not correspond to an existing data-object known to the **DDS::DataWriter** (p. 499).

Parameters:

key_holder *<<inout>>* *(p. 176)* a user data type specific key holder, whose `key` fields are filled by this operation. If **Foo** (p. 877) has no key, this method has no effect. This method will fail with **DDS::Retcode_BadParameter** (p. 1115) if `key_holder` is NULL.

handle *<<in>>* *(p. 175)* the `instance` whose key is to be retrieved. If **Foo** (p. 877) *has* a key, `handle` must represent a registered instance of type **Foo** (p. 877). Otherwise, this method will fail with **DDS::Retcode_BadParameter** (p. 1115). If **Foo** (p. 877) *has* a key and `handle` is **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), this method will fail with **DDS::Retcode_BadParameter** (p. 1115). This method will fail with **DDS::Retcode_BadParameter** (p. 1115) if `handle` is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-NotEnabled** (p. 1121).

See also:

DDS::TypedDataReader::get_key_value (p. 1365)

6.63.2.10 InstanceHandle_t DDS::DynamicDataWriter::lookup_-instance (DynamicData^ *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 Data Distribution Service is unable to provide an instance handle, RTI Data Distribution Service will return the special value HANDLE_NIL.

Parameters:

key_holder *<<in>>* *(p. 175)* a user data type specific key holder.

Returns:

the instance handle associated with this instance. If **Foo** (p. 877) has no key, this method has no effect and returns **DDS::InstanceHandle_-t::HANDLE_NIL** (p. 53)

6.64 DDS::Entity Class Reference

<<*interface*>> *(p. 175)* Abstract base class for all the DDS objects that support QoS policies, a listener, and a status condition.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::Entity::

Public Member Functions

- virtual void **enable** () override

  *Enables the **DDS::Entity** (p. 845).*

- virtual **StatusCondition^ get_statuscondition** () override

  *Allows access to the **DDS::StatusCondition** (p. 1183) associated with the **DDS::Entity** (p. 845).*

- virtual **StatusMask get_status_changes** () override

  *Retrieves the list of communication statuses in the **DDS::Entity** (p. 845) that are triggered.*

- virtual **InstanceHandle_t get_instance_handle** () override

  *Allows access to the **DDS::InstanceHandle_t** (p. 905) associated with the **DDS::Entity** (p. 845).*

6.64.1 Detailed Description

<<*interface*>> *(p. 175)* Abstract base class for all the DDS objects that support QoS policies, a listener, and a status condition.

All operations except for set_qos(), get_qos(), set_listener(), get_listener() and **enable()** (p. 848), may return the value **DDS::Retcode_NotEnabled** (p. 1121).

QoS:

QoS Policies (p. 260)

Status:

Status Kinds (p. 238)

Listener:

DDS::Listener (p. 952)

6.64.2 Abstract operations

Each derived entity provides the following operations specific to its role in RTI Data Distribution Service.

6.64.2.1 set_qos (abstract)

This operation sets the QoS policies of the **DDS::Entity** (p. 845).

This operation must be provided by each of the derived **DDS::Entity** (p. 845) classes (**DDS::DomainParticipant** (p. 577), **DDS::Topic** (p. 1258), **DDS::Publisher** (p. 1044), **DDS::DataWriter** (p. 499), **DDS::Subscriber** (p. 1201), and **DDS::DataReader** (p. 433)) so that the policies that are meaningful to each **DDS::Entity** (p. 845) can be set.

Precondition:

Certain policies are immutable (see **QoS Policies** (p. 260)): they can only be set at **DDS::Entity** (p. 845) creation time or before the entity is enabled. If `set_qos()` is invoked after the **DDS::Entity** (p. 845) is enabled and it attempts to change the value of an immutable policy, the operation will fail and return **DDS::Retcode_ImmutablePolicy** (p. 1118).
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_InconsistentPolicy** (p. 1119).
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. 1125).

Postcondition:

The existing set of policies is only changed if the `set_qos()` operation succeeds. This is indicated by a return code of DDS::Exception::RETCODE_OK. In all other cases, none of the policies are modified.

Each derived **DDS::Entity** (p. 845) class (**DDS::DomainParticipant** (p. 577), **DDS::Topic** (p. 1258), **DDS::Publisher** (p. 1044), **DDS::DataWriter** (p. 499), **DDS::Subscriber** (p. 1201), **DDS::DataReader** (p. 433)) has a corresponding special value of the

QoS (**DDS::DomainParticipantFactory::PARTICIPANT_QOS_DEFAULT** (p. 35), **DDS::DomainParticipant::PUBLISHER_QOS_DEFAULT** (p. 38), **DDS::DomainParticipant::SUBSCRIBER_QOS_DEFAULT** (p. 38), **DDS::DomainParticipant::TOPIC_QOS_DEFAULT** (p. 39), **DDS::Publisher::DATAWRITER_QOS_DEFAULT** (p. 80), **DDS::Subscriber::DATAREADER_QOS_DEFAULT** (p. 95)). This special value may be used as a parameter to the `set_qos` operation to indicate that the QoS of the **DDS::Entity** (p. 845) should be changed to match the current default QoS set in the **DDS::Entity** (p. 845)'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** (p. 845) has been modified to match the current default for the **DDS::Entity** (p. 845)'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. 235) : **DDS::Retcode_ImmutablePolicy** (p. 1118), or **DDS::Retcode_InconsistentPolicy** (p. 1119).

6.64.2.2 get_qos (abstract)

This operation allows access to the existing set of QoS policies for the **DDS::Entity** (p. 845). This operation must be provided by each of the derived **DDS::Entity** (p. 845) classes (**DDS::DomainParticipant** (p. 577), **DDS::Topic** (p. 1258), **DDS::Publisher** (p. 1044), **DDS::DataWriter** (p. 499), **DDS::Subscriber** (p. 1201), and **DDS::DataReader** (p. 433)), so that the policies that are meaningful to each **DDS::Entity** (p. 845) can be retrieved.

Possible error codes are **Standard Return Codes** (p. 235).

6.64.2.3 set_listener (abstract)

This operation installs a **DDS::Listener** (p. 952) on the **DDS::Entity** (p. 845). 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. 845) classes (**DDS::DomainParticipant** (p. 577), **DDS::Topic** (p. 1258), **DDS::Publisher** (p. 1044), **DDS::DataWriter** (p. 499), **DDS::Subscriber** (p. 1201), and **DDS::DataReader** (p. 433)), so that the listener is of the concrete type suitable to the particular **DDS::Entity** (p. 845).

It is permitted to use null as the value of the listener. The null listener behaves as if the `mask` is DDS::StatusMask::STATUS_MASK_NONE.

Postcondition:

Only one listener can be attached to each **DDS::Entity** (p. 845). 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.

6.64.2.4 get_listener (abstract)

This operation allows access to the existing **DDS::Listener** (p. 952) attached to the **DDS::Entity** (p. 845).

This operation must be provided by each of the derived **DDS::Entity** (p. 845) classes (**DDS::DomainParticipant** (p. 577), **DDS::Topic** (p. 1258), **DDS::Publisher** (p. 1044), **DDS::DataWriter** (p. 499), **DDS::Subscriber** (p. 1201), and **DDS::DataReader** (p. 433)) so that the listener is of the concrete type suitable to the particular **DDS::Entity** (p. 845).

If no listener is installed on the **DDS::Entity** (p. 845), this operation will return null.

6.64.3 Member Function Documentation

6.64.3.1 virtual void DDS::Entity::enable () [pure virtual]

Enables the **DDS::Entity** (p. 845).

This operation enables the **Entity** (p. 845). **Entity** (p. 845) objects can be created either enabled or disabled. This is controlled by the value of the **ENTITY_FACTORY** (p. 304) QoS policy on the corresponding factory for the **DDS::Entity** (p. 845).

By default, **ENTITY_FACTORY** (p. 304) is set so that it is not necessary to explicitly call **DDS::Entity::enable** (p. 848) on newly created entities.

The **DDS::Entity::enable** (p. 848) operation is idempotent. Calling enable on an already enabled **Entity** (p. 845) returns OK and has no effect.

If a **DDS::Entity** (p. 845) 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. 849)
- 'factory' operations
- **DDS::Entity::get_status_changes** (p. 850) 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_NotEnabled** (p. 1121).

It is legal to delete an **DDS::Entity** (p. 845) that has not been enabled by calling the proper operation on its factory.

Entities created from a factory that is disabled are created disabled, regardless of the setting of the **DDS::EntityFactoryQosPolicy** (p. 851).

Calling enable on an **Entity** (p. 845) whose factory is not enabled will fail and return **DDS::Retcode_PreconditionNotMet** (p. 1123).

If **DDS::EntityFactoryQosPolicy::autoenable_created_entities** (p. 852) is TRUE, the enable operation on a factory will automatically enable all entities created from that factory.

Listeners associated with an entity are not called until the entity is enabled.

Conditions associated with a disabled entity are "inactive," that is, they have a `trigger_value` == FALSE.

Exceptions:

One of the **Standard Return Codes** (p. 235), **Standard Return Codes** (p. 235) or **DDS::Retcode_PreconditionNotMet** (p. 1123).

Implemented in **DDS::DomainParticipant** (p. 646), **DDS::Publisher** (p. 1066), **DDS::DataWriter** (p. 520), **DDS::Subscriber** (p. 1223), **DDS::DataReader** (p. 456), and **DDS::Topic** (p. 1265).

6.64.3.2 virtual StatusCondition ^ DDS::Entity::get_statuscondition () [pure virtual]

Allows access to the **DDS::StatusCondition** (p. 1183) associated with the **DDS::Entity** (p. 845).

The returned condition can then be added to a **DDS::WaitSet** (p. 1411) so that the application can wait for specific status changes that affect the **DDS::Entity** (p. 845).

Returns:

the status condition associated with this entity.

Implemented in **DDS::DomainParticipant** (p. 647), **DDS::Publisher** (p. 1067), **DDS::DataWriter** (p. 521), **DDS::Subscriber** (p. 1224), **DDS::DataReader** (p. 457), and **DDS::Topic** (p. 1266).

6.64.3.3 virtual StatusMask DDS::Entity::get_status_changes () [pure virtual]

Retrieves the list of communication statuses in the **DDS::Entity** (p. 845) that are triggered.

That is, the list of statuses whose value has changed since the last time the application read the status using the get_*_status() method.

When the entity is first created or if the entity is not enabled, all communication statuses are in the "untriggered" state so the list returned by the get_status_changes operation will be empty.

The list of statuses returned by the get_status_changes operation refers to the status that are triggered on the **Entity** (p. 845) itself and does not include statuses that apply to contained entities.

Returns:

list of communication statuses in the **DDS::Entity** (p. 845) that are triggered.

See also:

Status Kinds (p. 238)

Implemented in **DDS::DomainParticipant** (p. 648), **DDS::Publisher** (p. 1067), **DDS::DataWriter** (p. 522), **DDS::Subscriber** (p. 1225), **DDS::DataReader** (p. 458), and **DDS::Topic** (p. 1266).

6.64.3.4 virtual InstanceHandle_t DDS::Entity::get_instance_handle () [pure virtual]

Allows access to the **DDS::InstanceHandle_t** (p. 905) associated with the **DDS::Entity** (p. 845).

This operation returns the **DDS::InstanceHandle_t** (p. 905) that represents the **DDS::Entity** (p. 845).

Returns:

the instance handle associated with this entity.

Implemented in **DDS::DomainParticipant** (p. 648), **DDS::Publisher** (p. 1068), **DDS::DataWriter** (p. 522), **DDS::Subscriber** (p. 1225), **DDS::DataReader** (p. 458), and **DDS::Topic** (p. 1267).

6.65 DDS::EntityFactoryQosPolicy Struct Reference

A QoS policy for all **DDS::Entity** (p. 845) types that can act as factories for one or more other **DDS::Entity** (p. 845) types.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_entityfactory_qos_policy_name** ()

  Stringified human-readable name for ***DDS::EntityFactoryQosPolicy*** *(p. 851).*

Properties

- System::Boolean **autoenable_created_entities** `[get, set]`

  Specifies whether the entity acting as a factory automatically enables the instances it creates.

6.65.1 Detailed Description

A QoS policy for all **DDS::Entity** (p. 845) types that can act as factories for one or more other **DDS::Entity** (p. 845) types.

Entity:

DDS::DomainParticipantFactory (p. 649), **DDS::DomainParticipant** (p. 577), **DDS::Publisher** (p. 1044), **DDS::Subscriber** (p. 1201)

Properties:

RxO (p. 268) = NO
Changeable (p. 269) = **YES** (p. 269)

6.65.2 Usage

This policy controls the behavior of the **DDS::Entity** (p. 845) as a factory for other entities. It controls whether or not child entities are created in the enabled state.

RTI Data Distribution Service 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. 433) is created in an enabled state, its existence is immediately propagated for discovery and the **DDS::DataReader** (p. 433) 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. 433), and thus, it may not be desireable for the **DDS::DataReader** (p. 433) 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.

6.65.3 Property Documentation

6.65.3.1 System:: Boolean DDS::EntityFactoryQosPolicy::autoenable_created_entities [get, set]

Specifies whether the entity acting as a factory automatically enables the instances it creates.

The setting of `autoenable_created_entities` to true indicates that the factory `create_<entity>` operation(s) will automatically invoke the **DDS::Entity::enable** (p. 848) operation each time a new **DDS::Entity** (p. 845) is created. Therefore, the **DDS::Entity** (p. 845) returned by `create_<entity>` will already be enabled. A setting of false indicates that the **DDS::Entity** (p. 845) will not be automatically enabled. Your application will need to call **DDS::Entity::enable** (p. 848) itself.

The default setting of `autoenable_created_entities` = true means that, by

default, it is not necessary to explicitly call **DDS::Entity::enable** (p. 848) on newly created entities.

[**default**] true

6.66 DDS::EntityNameQosPolicy Class Reference

Assigns a name to a **DDS::DomainParticipant** (p. 577). This name will be visible during the discovery process and in RTI tools to help you visualize and debug your system.

`#include <managed_infrastructure.h>`

Static Public Member Functions

^ static System::String^ **get_entityname_qos_policy_name** ()

*Stringified human-readable name for **DDS::EntityNameQosPolicy** (p. 854).*

Public Attributes

^ System::String^ **name**

The actual name string.

6.66.1 Detailed Description

Assigns a name to a **DDS::DomainParticipant** (p. 577). This name will be visible during the discovery process and in RTI tools to help you visualize and debug your system.

Entity:

DDS::DomainParticipant (p. 577)

Properties:

RxO (p. 268) = NO;
Changeable (p. 269) = **UNTIL ENABLE** (p. 269)

6.66.2 Usage

The name can only be 255 characters in length.

6.66.3 Member Data Documentation

6.66.3.1 System::String ^ DDS::EntityNameQosPolicy::name

The actual name string.

[**default**] "[ENTITY]"

[**range**] Null terminated string with length not exceeding 255. Can be NULL.

6.67 DDS::EnumMember Class Reference

A description of a member of an enumeration.

`#include <managed_typecode.h>`

Public Attributes

- ^ System::String^ **name**

  The name of the enumeration member.

- ^ System::Int32 **ordinal**

  The value associated the the enumeration member.

6.67.1 Detailed Description

A description of a member of an enumeration.

See also:

DDS::EnumMemberSeq (p. 857)
DDS::TypeCodeFactory::create_enum_tc (p. 1333)

6.67.2 Member Data Documentation

6.67.2.1 System::String ^ DDS::EnumMember::name

The name of the enumeration member.

Cannot be null.

6.67.2.2 System::Int32 DDS::EnumMember::ordinal

The value associated the the enumeration member.

6.68 DDS::EnumMemberSeq Class Reference

Defines a sequence of enumerator members.

`#include <managed_typecode.h>`

Inheritance diagram for DDS::EnumMemberSeq::

6.68.1 Detailed Description

Defines a sequence of enumerator members.

See also:

DDS::EnumMember (p. 856)
DDS::Sequence (p. 1163)
DDS::TypeCodeFactory::create_enum_tc (p. 1333)

6.69 DDS::EventQosPolicy Class Reference

Settings for event.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_event_qos_policy_name** ()
  Stringified human-readable name for ***DDS::EventQosPolicy*** *(*p. 858*).*

Public Attributes

- **ThreadSettings_t^ thread**
  Event thread QoS.
- System::Int32 **initial_count**
  The initial number of events.
- System::Int32 **max_count**
  The maximum number of events.

6.69.1 Detailed Description

Settings for event.

In a **DDS::DomainParticipant** (p. 577), 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. 577) 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. 577) from dynamically allocating memory for the event thread after initialization.

This QoS policy is an extension to the DDS standard.

Entity:

DDS::DomainParticipant (p. 577)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)

6.69.2 Member Data Documentation

6.69.2.1 ThreadSettings_t ^ DDS::EventQosPolicy::thread

Event thread QoS.

There is only one event thread.

Priority:

[**default**] The actual value depends on your architecture:

For Windows: -2

For Solaris: OS default priority

For Linux: OS default priority

For LynxOS: 13

For INTEGRITY: 80

For VxWorks: 110

For all others: OS default priority.

Stack Size:

[**default**] The actual value depends on your architecture:

For Windows: OS default stack size

For Solaris: OS default stack size

For Linux: OS default stack size

For LynxOS: 4∗16∗1024

For INTEGRITY: 4∗20∗1024

For VxWorks: 4∗16∗1024

For all others: OS default stack size.

Mask:

[**default**] mask = DDS::ThreadSettingsKind::THREAD_SETTINGS_FLOATING_POINT | DDS::ThreadSettingsKind::THREAD_SETTINGS_STDIO

6.69.2.2 System::Int32 DDS::EventQosPolicy::initial_count

The initial number of events.

[default] 256

[range] [1, 1 million], <= max_count

6.69.2.3 System::Int32 DDS::EventQosPolicy::max_count

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

[range] [1, 1 million] or DDS::LENGTH_UNLIMITED, >= initial_count

6.70 DDS::Exception Class Reference

Superclass of all exceptions thrown by the RTI Data Distribution Service API.

`#include <managed_exceptions.h>`

Inheritance diagram for DDS::Exception::

6.70.1 Detailed Description

Superclass of all exceptions thrown by the RTI Data Distribution Service API.

Applications are not expected to throw or extend this type, but to handle exceptions of its more-specific subclasses.

Examples:

HelloWorld_publisher.cpp, and **HelloWorld_subscriber.cpp**.

6.71 DDS::ExclusiveAreaQosPolicy Struct Reference

Configures multi-thread concurrency and deadlock prevention capabilities.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- ^ static System::String^ **get_exclusivearea_qos_policy_name** ()

  *Stringified human-readable name for **DDS::ExclusiveAreaQosPolicy*** *(p. 862).*

Properties

- ^ System::Boolean **use_shared_exclusive_area** `[get, set]`

  *Whether the **DDS::Entity*** *(p. 845)* *is protected by its own exclusive area or the shared exclusive area.*

6.71.1 Detailed Description

Configures multi-thread concurrency and deadlock prevention capabilities.

An "exclusive area" is an abstraction of a multi-thread-safe region. Each entity is protected by one and only one exclusive area, although a single exclusive area may be shared by multiple entities.

Conceptually, an exclusive area is a mutex or monitor with additional deadlock protection features. If a **DDS::Entity** (p. 845) has "entered" its exclusive area to perform a protected operation, no other **DDS::Entity** (p. 845) sharing the same exclusive area may enter it until the first **DDS::Entity** (p. 845) "exits" the exclusive area.

Entity:

DDS::Publisher (p. 1044), **DDS::Subscriber** (p. 1201)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)

See also:

DDS::Listener (p. 952)

6.71.2 Usage

Exclusive Areas (EAs) allow RTI Data Distribution Service to be multi-threaded while preventing deadlock in multi-threaded applications. EAs prevent a **DDS::DomainParticipant** (p. 577) object's internal threads from deadlocking with each other when executing internal code as well as when executing the code of user-registered listener callbacks.

Within an EA, all calls to the code protected by the EA are single threaded. Each **DDS::DomainParticipant** (p. 577), **DDS::Publisher** (p. 1044) and **DDS::Subscriber** (p. 1201) entity represents a separate EA. Thus all DataWriters of the same **Publisher** (p. 1044) and all DataReaders of the same **Subscriber** (p. 1201) share the EA of its parent. Note: this means that operations on the DataWriters of the same **Publisher** (p. 1044) and on the DataReaders of the same **Subscriber** (p. 1201) will be serialized, even when invoked from multiple concurrent application threads.

Within an EA, there are limitations on how code protected by a different EA can be accessed. For example, when received data is being processed by user code in the **DataReader** (p. 433) **Listener** (p. 952), within a **Subscriber** (p. 1201) EA, the user code may call the **DDS::TypedDataWriter::write** (p. 1376) operation of a **DataWriter** (p. 499) that is protected by the EA of its **Publisher** (p. 1044), so you can send data in the function called to process received data. However, you cannot create entities or call functions that are protected by the EA of the **DDS::DomainParticipant** (p. 577). See Chapter 4 in the RTI Data Distribution Service User's Manual for complete documentation on Exclusive Areas.

With this QoS policy, you can force a **DDS::Publisher** (p. 1044) or **DDS::Subscriber** (p. 1201) to share the same EA as its **DDS::DomainParticipant** (p. 577). Using this capability, the restriction of not being able to create entities in a **DataReader** (p. 433) Listener's on_data_available() callback is lifted. However, the tradeoff is that the application has reduced concurrency through the Entities that share an EA.

Note that the restrictions on calling methods in a different EA only exist for user code that is called in registered DDS Listeners by internal **DomainParticipant** (p. 577) threads. User code may call all RTI Data Distribution Service functions for any DDS Entities from their own threads at any time.

6.71.3 Property Documentation

6.71.3.1 System:: Boolean DDS::ExclusiveAreaQosPolicy::use_shared_exclusive_area [get, set]

Whether the **DDS::Entity** (p. 845) is protected by its own exclusive area or the shared exclusive area.

All writers belonging to the same **DDS::Publisher** (p. 1044) are protected by the same exclusive area as the **DDS::Publisher** (p. 1044) itself. The same is true of all readers belonging to the same **DDS::Subscriber** (p. 1201). Typically, the publishers and subscribers themselves do not share their exclusive areas with each other; each has its own. This configuration maximizes the concurrency of the system because independent readers and writers do not need to take the same mutexes in order to operate. However, it places some restrictions on the operations that may be invoked from within listener callbacks because of the possibility of a deadlock. See the **DDS::Listener** (p. 952) documentation for more details.

If this field is set to false, the default more concurrent behavior will be used. In the event that this behavior is insufficiently flexible for your application, you may set this value to true. In that case, the **DDS::Subscriber** (p. 1201) or **DDS::Publisher** (p. 1044) in question, and all of the readers or writers (as appropriate) created from it, will share a global exclusive area. This global exclusive area is shared by all entities whose value for this QoS field is true. By sharing the same exclusive area across a larger number of entities, the concurrency of the system will be decreased; however, some of the callback restrictions will be relaxed.

[**default**] false

6.72 DDS::FloatSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < System::Single >.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::FloatSeq::

Public Member Functions

- ^ **FloatSeq** ()

  Constructs an empty sequence of floats with an initial maximum of zero.

- ^ **FloatSeq** (System::Int32 max)

  Constructs an empty sequence of floats with the given initial maximum.

- ^ **FloatSeq** (**FloatSeq**^ floats)

  Constructs a new sequence containing the given floats.

6.72.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < System::Single >.

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

System::Single
DDS::Sequence (p. 1163)

6.72.2 Constructor & Destructor Documentation

6.72.2.1 DDS::FloatSeq::FloatSeq () `[inline]`

Constructs an empty sequence of floats with an initial maximum of zero.

6.72.2.2 DDS::FloatSeq::FloatSeq (System::Int32 *max*) `[inline]`

Constructs an empty sequence of floats with the given initial maximum.

6.72.2.3 DDS::FloatSeq::FloatSeq (FloatSeq^ *floats*) [inline]

Constructs a new sequence containing the given floats.

Parameters:

floats the initial contents of this sequence

6.73 DDS::FlowController Class Reference

<<interface>> (p. 175) A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous **DDS::DataWriter** (p. 499) instances are allowed to write data.

`#include <managed_flowcontroller.h>`

Public Member Functions

- ^ void **set_property** (**FlowControllerProperty_t**^ prop)

  *Sets the **DDS::FlowController** (p. 867) property.*

- ^ void **get_property** (**FlowControllerProperty_t**^ prop)

  *Gets the **DDS::FlowController** (p. 867) property.*

- ^ void **trigger_flow** ()

  *Provides an external trigger to the **DDS::FlowController** (p. 867).*

- ^ System::String^ **get_name** ()

  *Returns the name of the **DDS::FlowController** (p. 867).*

- ^ **DomainParticipant**^ **get_participant** ()

  *Returns the **DDS::DomainParticipant** (p. 577) to which the **DDS::FlowController** (p. 867) belongs.*

Properties

- ^ static System::String^ **DEFAULT_FLOW_CONTROLLER_NAME** `[get]`

  [**default**] *Special value of **DDS::PublishModeQosPolicy::flow_controller_name** (p. 1079) that refers to the built-in default flow controller.*

- ^ static System::String^ **FIXED_RATE_FLOW_CONTROLLER_NAME** `[get]`

  *Special value of **DDS::PublishModeQosPolicy::flow_controller_name** (p. 1079) that refers to the built-in fixed-rate flow controller.*

- ^ static System::String^ **ON_DEMAND_FLOW_CONTROLLER_NAME** `[get]`

  *Special value of **DDS::PublishModeQosPolicy::flow_controller_name** (p. 1079) that refers to the built-in on-demand flow controller.*

6.73.1 Detailed Description

<<*interface*>> *(p. 175)* A flow controller is the object responsible for shaping the network traffic by determining when attached asynchronous **DDS::DataWriter** (p. 499) instances are allowed to write data.

QoS:

DDS::FlowControllerProperty_t (p. 871)

6.73.2 Member Function Documentation

6.73.2.1 void DDS::FlowController::set_property (FlowControllerProperty_t^ *prop*)

Sets the **DDS::FlowController** (p. 867) property.

This operation modifies the property of the **DDS::FlowController** (p. 867).

Once a **DDS::FlowController** (p. 867) has been instantiated, only the **DDS::FlowControllerProperty_t::token_bucket** (p. 872) can be changed. The **DDS::FlowControllerProperty_t::scheduling_policy** (p. 872) is immutable.

A new **DDS::FlowControllerTokenBucketProperty_t::period** (p. 875) only takes effect at the next scheduled token distribution time (as determined by its previous value).

Parameters:

prop <<*in*>> *(p. 175)* The new **DDS::FlowControllerProperty_t** (p. 871). Property must be consistent. Immutable fields cannot be changed after **DDS::FlowController** (p. 867) has been created. The special value DDS::FLOW_CONTROLLER_PROPERTY_DEFAULT can be used to indicate that the property of the **DDS::FlowController** (p. 867) should be changed to match the current default **DDS::FlowControllerProperty_t** (p. 871) set in the **DDS::DomainParticipant** (p. 577). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_ImmutablePolicy** (p. 1118), or **DDS::Retcode_InconsistentPolicy** (p. 1119).

See also:

DDS::FlowControllerProperty_t (p. 871) for rules on consistency among property values.

6.73.2.2 void DDS::FlowController::get_property (FlowControllerProperty_t^ *prop*)

Gets the **DDS::FlowController** (p. 867) property.

Parameters:

prop *<<in>>* *(p. 175)* **DDS::FlowController** (p. 867) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.73.2.3 void DDS::FlowController::trigger_flow ()

Provides an external trigger to the **DDS::FlowController** (p. 867).

Typically, a **DDS::FlowController** (p. 867) 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. 875) property setting.

This function provides an additional, external trigger to the **DDS::FlowController** (p. 867). This trigger adds **DDS::FlowControllerTokenBucketProperty_t::tokens_added_per_-period** (p. 874) tokens each time it is called (subject to the other property settings of the **DDS::FlowController** (p. 867)).

An *on-demand* **DDS::FlowController** (p. 867) can be created with a **DDS::Duration_t::DURATION_INFINITE** (p. 253) as **DDS::FlowControllerTokenBucketProperty_t::period** (p. 875), in which case the only trigger source is external (i.e. the **DDS::FlowController** (p. 867) is solely triggered by the user on demand).

DDS::FlowController::trigger_flow (p. 869) can be called on both strict *on-demand* **DDS::FlowController** (p. 867) and hybrid **DDS::FlowController** (p. 867) (internally and externally triggered).

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.73.2.4 System::String ^ DDS::FlowController::get_name ()

Returns the name of the **DDS::FlowController** (p. 867).

Returns:

The name of the **DDS::FlowController** (p. 867).

6.73.2.5 DomainParticipant ^ DDS::FlowController::get_participant ()

Returns the **DDS::DomainParticipant** (p. 577) to which the **DDS::FlowController** (p. 867) belongs.

Returns:

The **DDS::DomainParticipant** (p. 577) to which the **DDS::FlowController** (p. 867) belongs.

6.74 DDS::FlowControllerProperty_t Class Reference

Determines the flow control characteristics of the **DDS::FlowController** (p. 867).

`#include <managed_flowcontroller.h>`

Public Attributes

- **FlowControllerSchedulingPolicy scheduling_policy**

  Scheduling policy.

- **FlowControllerTokenBucketProperty_t token_bucket**

  Settings for the token bucket.

6.74.1 Detailed Description

Determines the flow control characteristics of the **DDS::FlowController** (p. 867).

The flow control characteristics shape the network traffic by determining how often and in what order associated asynchronous **DDS::DataWriter** (p. 499) instances are serviced and how much data they are allowed to send.

Note that these settings apply directly to the **DDS::FlowController** (p. 867), and does not depend on the number of **DDS::DataWriter** (p. 499) instances the **DDS::FlowController** (p. 867) is servicing. For instance, the specified flow rate does *not* double simply because two **DDS::DataWriter** (p. 499) instances are waiting to write.

Entity:

DDS::FlowController (p. 867)

Properties:

RxO (p. 268) = N/A

Changeable (p. 269) = **NO** (p. 269) for **DDS::FlowControllerProperty_t::scheduling_policy** (p. 872), **YES** (p. 269) for **DDS::FlowControllerProperty_t::token_bucket** (p. 872). However, the special value of **DDS::Duration_t::DURATION_INFINITE** (p. 253) as **DDS::FlowControllerTokenBucketProperty_t::period** (p. 875) is

strictly used to create an *on-demand* **DDS::FlowController** (p. 867). The token period cannot toggle from an infinite to finite value (or vice versa). It can, however, change from one finite value to another.

6.74.2 Member Data Documentation

6.74.2.1 FlowControllerSchedulingPolicy DDS::FlowControllerProperty_t::scheduling_policy

Scheduling policy.

Determines the scheduling policy for servicing the **DDS::DataWriter** (p. 499) instances associated with the **DDS::FlowController** (p. 867).

[default] idref_FlowControllerSchedulingPolicy_EDF_FLOW_CONTROLLER_SCHED_POLICY

6.74.2.2 FlowControllerTokenBucketProperty_t DDS::FlowControllerProperty_t::token_bucket

Settings for the token bucket.

6.75 DDS::FlowControllerTokenBucketProperty_t Struct Reference

DDS::FlowController (p. 867) uses the popular token bucket approach for open loop network flow control. The flow control characteristics are determined by the token bucket properties.

`#include <managed_flowcontroller.h>`

Public Attributes

- System::Int32 **max_tokens**

  Maximum number of tokens than can accumulate in the token bucket.

- System::Int32 **tokens_added_per_period**

  The number of tokens added to the token bucket per specified period.

- System::Int32 **tokens_leaked_per_period**

  The number of tokens removed from the token bucket per specified period.

- **Duration_t period**

  Period for adding tokens to and removing tokens from the bucket.

- System::Int32 **bytes_per_token**

  Maximum number of bytes allowed to send for each token available.

6.75.1 Detailed Description

DDS::FlowController (p. 867) 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. 867)

Properties:

RxO (p. 268) = N/A

Changeable (p. 269) = **YES** (p. 269). However, the special value of **DDS::Duration_t::DURATION_INFINITE** (p. 253) as **DDS::FlowControllerTokenBucketProperty_t::period** (p. 875) is strictly used to create an *on-demand* **DDS::FlowController** (p. 867). The token period cannot toggle from an infinite to finite value (or vice versa). It can, however, change from one finite value to another.

6.75.2 Member Data Documentation

6.75.2.1 System::Int32 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. 876), determines the maximum allowable data burst.

Use DDS::LENGTH_UNLIMITED to allow accumulation of an unlimited amount of tokens (and therefore potentially an unlimited burst size).

[**default**] DDS::LENGTH_UNLIMITED

6.75.2.2 System::Int32 DDS::FlowControllerTokenBucketProperty_t::tokens_added_per_period

The number of tokens added to the token bucket per specified period.

DDS::FlowController (p. 867) 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. 499) instances based on the **DDS::FlowControllerProperty_t::scheduling_policy** (p. 872).

Use DDS::LENGTH_UNLIMITED to add the maximum number of tokens allowed by **DDS::FlowControllerTokenBucketProperty_t::max_tokens** (p. 874).

[**default**] DDS::LENGTH_UNLIMITED

6.75.2.3 System::Int32 DDS::FlowControllerTokenBucketProperty_t::tokens_leaked_per_period

The number of tokens removed from the token bucket per specified period.

DDS::FlowController (p. 867) 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 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

6.75.2.4 Duration_t DDS::FlowControllerTokenBucketProperty_t::period

Period for adding tokens to and removing tokens from the bucket.

DDS::FlowController (p. 867) 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_t::DURATION_INFINITE** (p. 253) can be used to create an *on-demand* **DDS::FlowController** (p. 867), for which tokens are no longer replenished periodically. Instead, tokens must be added explicitly by calling **DDS::FlowController::trigger_flow** (p. 869). This external trigger adds **DDS::FlowControllerTokenBucketProperty_t::tokens_added_per_period** (p. 874) tokens each time it is called (subject to the other property settings).

[**default**] 1 second

[**range**] [0,1 year] or **DDS::Duration_t::DURATION_INFINITE** (p. 253)

6.75.2.5 System::Int32 DDS::FlowControllerTokenBucketProperty_t::bytes_per_token

Maximum number of bytes allowed to send for each token available.

DDS::FlowController (p. 867) 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. 499). That is, in cases where **DDS::FlowControllerTokenBucketProperty_t::bytes_per_token** (p. 876) 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. 872)).

Where fragmentation is required, the fragment size will be **DDS::FlowControllerTokenBucketProperty_t::bytes_per_token** (p. 876) or the minimum largest message size across all transports installed with the **DDS::DataWriter** (p. 499), whichever is less.

Use DDS::LENGTH_UNLIMITED 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. 499) 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

[**range**] [1024,DDS::LENGTH_UNLIMITED]

6.76 Foo Struct Reference

A representative user-defined data type.

`#include <managed_topic.h>`

Inheritance diagram for Foo::

6.76.1 Detailed Description

A representative user-defined data type.

Foo (p. 877) represents a user-defined data-type that is intended to be distributed using DDS.

The type **Foo** (p. 877) is usually defined using IDL syntax and placed in a ".idl" file that is then processed using **rtiddsgen** (p. 196). The **rtiddsgen** (p. 196) utility generates the helper classes **DDS::Sequence** (p. 1163) 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 **DDS::TypedDataReader** (p. 1338) and **DDS::TypedDataWriter** (p. 1368) types that allow the application to send and receive data of this type.

See also:

DDS::Sequence (p. 1163), **DDS::TypedDataWriter** (p. 1368), **DDS::TypedDataReader** (p. 1338), **FooTypeSupport** (p. 884), **rtiddsgen** (p. 196)

6.77 FooDataReader Class Reference

*<<**interface**>> (p. 175) <<**generic**>> (p. 175)* User data type-specific data reader.

`#include <FooSupport.h>`

Inheritance diagram for FooDataReader::

6.77.1 Detailed Description

*<<**interface**>> (p. 175) <<**generic**>> (p. 175)* 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. 433)
Foo (p. 877)
DDS::TypedDataWriter (p. 1368)
rtiddsgen (p. 196)

A reader for the **Foo** (p. 877) type.

6.78 FooDataWriter Class Reference

<<*interface*>> *(p. 175)* <<*generic*>> *(p. 175)* User data type specific data writer.

`#include <FooSupport.h>`

Inheritance diagram for FooDataWriter::

6.78.1 Detailed Description

<<*interface*>> *(p. 175)* <<*generic*>> *(p. 175)* 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. 499)
Foo (p. 877)
DDS::TypedDataReader (p. 1338)
rtiddsgen (p. 196)

A writer for the **Foo** (p. 877) user type.

6.79 FooSeq Class Reference

*<<**interface**>> (p. 175) <<**generic**>> (p. 175)* A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as `Foo (p. 877)`.

`#include <managed_sequence.h>`

Inheritance diagram for FooSeq::

Public Member Functions

- **FooSeq** ()

  Create a sequence with a maximum of 0.

- **FooSeq** (System::Int32 new_max)

  Create a sequence with the given maximum.

- **FooSeq** (**FooSeq**^ src)

  Create a sequence by copying from an existing sequence.

6.79.1 Detailed Description

*<<**interface**>> (p. 175) <<**generic**>> (p. 175)* A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as `Foo (p. 877)`.

For users who define data types in OMG IDL, this type corresponds to the IDL express `sequence<Foo (p. 877)>`.

For any user-data type `Foo (p. 877)` that an application defines for the purpose of data-distribution with RTI Data Distribution Service, a `FooSeq (p. 880)` is generated. We refer to an IDL `sequence<Foo (p. 877)> as FooSeq (p. 880)`.

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 **DDS::Sequence::maximum** (p. 1172) operation.

- The 'length' represents the actual number of elements it currently holds. It is returned by the **DDS::Sequence::length** (p. 1171) operation.

^ The 'owned' flag represents whether the sequence owns the underlying buffer. It is returned by the **DDS::Sequence::has_ownership** (p. 1173) 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 == true, the sequence has ownership on the buffer. It is then responsible for destroying the buffer when the sequence is destroyed.
- If the owned == false, 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 == true

See also:

DDS::TypedDataWriter (p. 1368), **DDS::TypedDataReader** (p. 1338), **FooTypeSupport** (p. 884), **rtiddsgen** (p. 196)

6.79.2 Constructor & Destructor Documentation

6.79.2.1 FooSeq::FooSeq ()

Create a sequence with a maximum of 0.

This is a constructor for the sequence. The constructor will allocate no memory.

This constructor will be used when the application creates a sequence using one of the following:

In C#:

```
FooSeq my_seq = new FooSeq();
```

In C++/CLI:

```
FooSeq^ my_seq = gcnew FooSeq();
```

Postcondition:

maximum == 0
length == 0
owned == true,

6.79.2.2 FooSeq::FooSeq (System::Int32 *new_max*)

Create a sequence with the given maximum.

This is a constructor for the sequence. The constructor will automatically allocate memory to hold new_max elements of type **Foo** (p. 877).

This constructor will be used when the application creates a sequence using one of the following:

In C#:

```
FooSeq my_seq = new FooSeq(5);
```

In C++/CLI:

```
FooSeq^ my_seq = gcnew FooSeq(5);
```

Postcondition:

maximum == new_max
length == 0
owned == true,

Parameters:

new_max Must be >= 0. Otherwise the sequence will be initialized to a new_max=0.

6.79.2.3 FooSeq::FooSeq (FooSeq^ *src*)

Create a sequence by copying from an existing sequence.

This is a constructor for the sequence. The constructor will automatically allocate memory to hold foo_seq::maximum() elements of type **Foo** (p. 877) and will copy the current contents of foo_seq into the new sequence.

This constructor will be used when the application creates a sequence using one of the following:

In C#:

```
FooSeq my_seq = new FooSeq(foo_seq);
```

In C++/CLI:

```
FooSeq^ my_seq = gcnew FooSeq(foo_seq);
```

Postcondition:

this::maximum == foo_seq::maximum
this::length == foo_seq::length
this[i] == foo_seq[i] for 0 <= i < foo_seq::length
this::owned == true

Note:

If the pre-conditions are not met, the constructor will initialize the new sequence to a maximum of zero.

6.80 FooTypeSupport Class Reference

*<<**interface**>> (p. 175) <<**generic**>> (p. 175)* User data type specific interface.

`#include <managed_topic.h>`

Inheritance diagram for FooTypeSupport::

Static Public Member Functions

- static System::String^ **get_type_name** ()

  Get the default name for this type.

- static void **register_type** (**DDS::DomainParticipant**^ participant, System::String^ type_name)

  Allows an application to communicate to RTI Data Distribution Service the existence of a data type.

- static void **unregister_type** (**DDS::DomainParticipant**^ participant, System::String^ type_name)

  Allows an application to unregister a data type from RTI Data Distribution Service. After calling unregister_type, no further communication using that type is possible.

- static **Foo**^ **create_data** ()

  <<**eXtension**>> (p. 174) *Create a data type and initialize it.*

- static void **delete_data** (**Foo**^ a_data)

  <<**eXtension**>> (p. 174) *Destroy a user data type instance.*

- static void **print_data** (**Foo**^ a_data)

  <<**eXtension**>> (p. 174) *Print value of data type to standard out.*

- static void **copy_data** (**Foo**^ dst_data, **Foo**^ src_data)

  <<**eXtension**>> (p. 174) *Copy data type.*

6.80.1 Detailed Description

*<<**interface**>> (p. 175) <<**generic**>> (p. 175)* 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:

rtiddsgen (p. 196)

6.80.2 Member Function Documentation

6.80.2.1 static System::String ^ FooTypeSupport::get_type_name () [static]

Get the default name for this type.

Can be used for calling **FooTypeSupport::register_type** (p. 885) or creating **DDS::Topic** (p. 1258)

Returns:

default name for this type

See also:

FooTypeSupport::register_type (p. 885)
DDS::DomainParticipant::create_topic (p. 621)

6.80.2.2 static void FooTypeSupport::register_type (DDS::DomainParticipant^ *participant*, System::String^ *type_name*) [static]

Allows an application to communicate to RTI Data Distribution Service the existence of a data type.

The *generated* implementation of the operation embeds all the knowledge that has to be communicated to the middleware in order to make it able to manage the contents of data of that type. This includes in particular the key definition that will allow RTI Data Distribution Service to distinguish different instances of the same type.

The same **DDS::TypeSupport** (p. 1385) can be registered multiple times with a **DDS::DomainParticipant** (p. 577) using the same or different values for the `type_name`. If `register_type` is called multiple times on the same **DDS::TypeSupport** (p. 1385) with the same **DDS::DomainParticipant** (p. 577) and `type_name`, the second (and subsequent) registrations are ignored by the operation fails with DDS::Exception::RETCODE_OK.

Precondition:

Cannot use the same `type_name` to register two different **DDS::TypeSupport** (p. 1385) with the same **DDS::DomainParticipant** (p. 577), or else the operation will fail and **DDS::Retcode_PreconditionNotMet** (p. 1123) will be returned.

Parameters:

participant *<<in>>* *(p. 175)* the **DDS::DomainParticipant** (p. 577) to register the data type `Foo` (p. 877) with. Cannot be NULL.

type_name *<<in>>* *(p. 175)* the type name under with the data type `Foo` (p. 877) is registered with the participant; this type name is used when creating a new **DDS::Topic** (p. 1258). (See **DDS::DomainParticipant::create_topic** (p. 621).) The name may not be NULL or longer than 255 characters.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_PreconditionNotMet** (p. 1123) or **DDS::Retcode_OutOfResources** (p. 1122).

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. 621)

6.80.2.3 static void FooTypeSupport::unregister_type (DDS::DomainParticipant^ *participant*, System::String^ *type_name*) [static]

Allows an application to unregister a data type from RTI Data Distribution Service. 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 Data Distribution Service. No further communication using that type is possible.

Precondition:

A type with `type_name` is registered with the participant and all **DDS::Topic** (p. 1258) objects referencing the type have been destroyed.

If the type is not registered with the participant, or if any **DDS::Topic** (p. 1258) is associated with the type, the operation will fail with **DDS::Retcode_Error** (p. 1116).

Postcondition:

All information about the type is removed from RTI Data Distribution Service. No further communication using this type is possible.

Parameters:

participant *<<in>>* *(p. 175)* the **DDS::DomainParticipant** (p. 577) to unregister the data type `Foo` (p. 877) from. Cannot be NULL.

type_name *<<in>>* *(p. 175)* the type name under with the data type `Foo` (p. 877) is registered with the participant. The name should match a name that has been previously used to register a type with the participant. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_BadParameter** (p. 1115) or **DDS::Retcode_Error** (p. 1116)

MT Safety:

SAFE.

See also:

FooTypeSupport::register_type (p. 885)

6.80.2.4 static Foo ^ FooTypeSupport::create_data () [static]

<<eXtension>> *(p. 174)* Create a data type and initialize it.

The *generated* implementation of the operation knows how to instantiate a data type and initialize it properly.

All memory for the type is deeply allocated.

Returns:

newly created data type

See also:

FooTypeSupport::delete_data (p. 888)

6.80.2.5 static void FooTypeSupport::delete_data (Foo^ *a_data*) [static]

<<eXtension>> (p. 174) Destroy a user data type instance.

The *generated* implementation of the operation knows how to destroy a data type and return all resources.

Parameters:

a_data *<<in>> (p. 175)* Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

FooTypeSupport::create_data (p. 887)

6.80.2.6 static void FooTypeSupport::print_data (Foo^ *a_data*) [static]

<<eXtension>> (p. 174) Print value of data type to standard out.

The *generated* implementation of the operation knows how to print value of a data type.

Parameters:

a_data *<<in>> (p. 175)* Data type to be printed.

6.80.2.7 static void FooTypeSupport::copy_data (Foo^ *dst_data*, Foo^ *src_data*) [static]

<<eXtension>> (p. 174) Copy data type.

The *generated* implementation of the operation knows how to copy value of a data type.

Parameters:

dst_data *<<inout>> (p. 176)* Data type to copy value to. Cannot be NULL.

src_data *<<in>> (p. 175)* Data type to copy value from. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.81 DDS::GroupDataQosPolicy Class Reference

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 42) during discovery.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_groupdata_qos_policy_name** ()

  *Stringified human-readable name for **DDS::GroupDataQosPolicy** (p. 890).*

Public Attributes

- **ByteSeq^ value**

  a sequence of octets

6.81.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 42) during discovery.

Entity:

DDS::Publisher (p. 1044), **DDS::Subscriber** (p. 1201)

Properties:

RxO (p. 268) = NO
Changeable (p. 269) = **YES** (p. 269)

See also:

DDS::DomainParticipant::get_builtin_subscriber (p. 632)

6.81.2 Usage

The additional information is attached to a **DDS::Publisher** (p. 1044) or **DDS::Subscriber** (p. 1201). This extra data is not used by RTI Data Distribution Service itself. When a remote application discovers the **DDS::Publisher**

(p. 1044) or **DDS::Subscriber** (p. 1201), 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. 1276) and **DDS::UserDataQosPolicy** (p. 1403), 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. 461), **DDS::DataWriterListener** (p. 524) and operations such as **DDS::DomainParticipant::ignore_publication** (p. 635) and **DDS::DomainParticipant::ignore_subscription** (p. 636), 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. 1008), 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 Data Distribution Service stores the data placed in this policy in pre-allocated pools. It is therefore necessary to configure RTI Data Distribution Service 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. 701) and **DDS::DomainParticipantResourceLimitsQosPolicy::subscriber_group_data_max_length** (p. 702).

6.81.3 Member Data Documentation

6.81.3.1 ByteSeq ^ DDS::GroupDataQosPolicy::value

a sequence of octets

[**default**] Empty (zero-sized)

[**range**] Octet sequence of length [0,max_length]

6.82 DDS::GuardCondition Class Reference

<<*interface*>> *(p. 175)* A specific **DDS::Condition** (p. 408) whose `trigger_value` is completely under the control of the application.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::GuardCondition::

Public Member Functions

- virtual System::Boolean **get_trigger_value** () override

  Retrieve the `trigger_value`.

- virtual void **set_trigger_value** (System::Boolean value)

  Set the guard condition trigger value.

- **GuardCondition** ()

  No argument constructor.

- ~**GuardCondition** ()

  Destructor.

6.82.1 Detailed Description

<<*interface*>> *(p. 175)* A specific **DDS::Condition** (p. 408) whose `trigger_value` is completely under the control of the application.

The **DDS::GuardCondition** (p. 892) provides a way for an application to manually wake up a **DDS::WaitSet** (p. 1411). This is accomplished by attaching the **DDS::GuardCondition** (p. 892) to the **DDS::WaitSet** (p. 1411) and then setting the `trigger_value` by means of the **DDS::GuardCondition::set_trigger_value** (p. 893) operation.

See also:

DDS::WaitSet (p. 1411)

6.82.2 Constructor & Destructor Documentation

6.82.2.1 DDS::GuardCondition::GuardCondition ()

No argument constructor.

Construct a new guard condition on the heap.

Returns:

A new condition with trigger value false, or null if a condition could not be allocated.

6.82.2.2 DDS::GuardCondition::~GuardCondition ()

Destructor.

Releases the resources associated with this object.

Calling this method multiple times on the same object is safe; subsequent deletions will have no effect.

6.82.3 Member Function Documentation

6.82.3.1 virtual System::Boolean DDS::GuardCondition::get_trigger_value () [override, virtual]

Retrieve the `trigger_value`.

Returns:

the trigger value.

Implements **DDS::Condition** (p. 408).

6.82.3.2 virtual void DDS::GuardCondition::set_trigger_value (System::Boolean *value*) [virtual]

Set the guard condition trigger value.

Parameters:

value *<<in>>* *(p. 175)* the new trigger value.

6.83 DDS::GUID_t Struct Reference

Type for *GUID* (Global Unique Identifier) representation.

`#include <managed_infrastructure.h>`

Public Attributes

- System::Byte **value_01**

  A 16 byte array containing the GUID value.

- System::Byte **value_02**

  A 16 byte array containing the GUID value.

- System::Byte **value_03**

  A 16 byte array containing the GUID value.

- System::Byte **value_04**

  A 16 byte array containing the GUID value.

- System::Byte **value_05**

  A 16 byte array containing the GUID value.

- System::Byte **value_06**

  A 16 byte array containing the GUID value.

- System::Byte **value_07**

  A 16 byte array containing the GUID value.

- System::Byte **value_08**

  A 16 byte array containing the GUID value.

- System::Byte **value_09**

  A 16 byte array containing the GUID value.

- System::Byte **value_10**

  A 16 byte array containing the GUID value.

- System::Byte **value_11**

  A 16 byte array containing the GUID value.

- System::Byte **value_12**

  A 16 byte array containing the GUID value.

- System::Byte **value_13**

  A 16 byte array containing the GUID value.

- System::Byte **value_14**

  A 16 byte array containing the GUID value.

- System::Byte **value_15**

  A 16 byte array containing the GUID value.

- System::Byte **value_16**

  A 16 byte array containing the GUID value.

Properties

- static **GUID_t GUID_UNKNOWN** `[get]`

  Unknown GUID.

- static **GUID_t GUID_AUTO** `[get]`

  Indicates that RTI Data Distribution Service should choose an appropriate virtual GUID.

6.83.1 Detailed Description

Type for *GUID* (Global Unique Identifier) representation.

Represents a 128 bit GUID.

6.83.2 Member Data Documentation

6.83.2.1 System::Byte DDS::GUID_t::value_01

A 16 byte array containing the GUID value.

6.83.2.2 System::Byte DDS::GUID_t::value_02

A 16 byte array containing the GUID value.

6.83.2.3 System::Byte DDS::GUID_t::value_03

A 16 byte array containing the GUID value.

6.83.2.4 System::Byte DDS::GUID_t::value_04

A 16 byte array containing the GUID value.

6.83.2.5 System::Byte DDS::GUID_t::value_05

A 16 byte array containing the GUID value.

6.83.2.6 System::Byte DDS::GUID_t::value_06

A 16 byte array containing the GUID value.

6.83.2.7 System::Byte DDS::GUID_t::value_07

A 16 byte array containing the GUID value.

6.83.2.8 System::Byte DDS::GUID_t::value_08

A 16 byte array containing the GUID value.

6.83.2.9 System::Byte DDS::GUID_t::value_09

A 16 byte array containing the GUID value.

6.83.2.10 System::Byte DDS::GUID_t::value_10

A 16 byte array containing the GUID value.

6.83.2.11 System::Byte DDS::GUID_t::value_11

A 16 byte array containing the GUID value.

6.83.2.12 System::Byte DDS::GUID_t::value_12

A 16 byte array containing the GUID value.

6.83.2.13 System::Byte DDS::GUID_t::value_13

A 16 byte array containing the GUID value.

6.83.2.14 System::Byte DDS::GUID_t::value_14

A 16 byte array containing the GUID value.

6.83.2.15 System::Byte DDS::GUID_t::value_15

A 16 byte array containing the GUID value.

6.83.2.16 System::Byte DDS::GUID_t::value_16

A 16 byte array containing the GUID value.

6.84 DDS::HistoryQosPolicy Struct Reference

Specifies the behavior of RTI Data Distribution Service 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.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- ^ static System::String^ **get_history_qos_policy_name** ()

  Stringified human-readable name for ***DDS::HistoryQosPolicy*** *(*p. 898*).*

Public Attributes

- ^ **HistoryQosPolicyKind kind**

  Specifies the kind of history to be kept.

- ^ System::Int32 **depth**

  Specifies the number of samples to be kept, when the `kind` *is DDS::HistoryQosPolicyKind::KEEP_LAST_HISTORY_QOS.*

- ^ **RefilterQosPolicyKind refilter**

  <<**eXtension**>> (p. 174) *Specifies how a writer should handle previously written samples to a new reader.*

6.84.1 Detailed Description

Specifies the behavior of RTI Data Distribution Service in the case where the value of a sample changes (one or more times) before it can be successfully communicated to one or more existing subscribers.

This QoS policy specifies how much data must to stored by RTI Data Distribution Service for a **DDS::DataWriter** (p. 499) or **DDS::DataReader** (p. 433). It controls whether RTI Data Distribution Service 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. 499) on behalf of existing **DDS::DataReader** (p. 433) entities. The behavior with regards to a

DDS::DataReader (p. 433) entities discovered after a sample is written is controlled by the **DURABILITY** (p. 276) policy.

On the subscribing side, this QoS policy controls the samples that should be maintained until the application "takes" them from RTI Data Distribution Service.

Entity:

DDS::Topic (p. 1258), **DDS::DataReader** (p. 433), **DDS::DataWriter** (p. 499)

Properties:

RxO (p. 268) = NO
Changeable (p. 269) = **UNTIL ENABLE** (p. 269)

See also:

DDS::ReliabilityQosPolicy (p. 1094)
DDS::HistoryQosPolicy (p. 898)

6.84.2 Usage

This policy controls the behavior of RTI Data Distribution Service when the value of an instance changes before it is finally communicated to **DDS::DataReader** (p. 433) entities.

When a **DDS::DataWriter** (p. 499) sends data, or a **DDS::DataReader** (p. 433) 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. 1094) by controlling whether RTI Data Distribution Service guarantees that *all* of the sent data is received (DDS::HistoryQosPolicyKind::KEEP_ALL_HISTORY_QOS) or if only the last N data values sent are guaranteed to be received (DDS::HistoryQosPolicyKind::KEEP_ALL_HISTORY_QOS)–this is a reduced level of reliability.

The amount of data that is sent to new DataReaders who have configured their **DDS::DurabilityQosPolicy** (p. 709) to receive previously published data is also controlled 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. 1109).

If `kind` is DDS::HistoryQosPolicyKind::KEEP_LAST_HISTORY_QOS (the default), then RTI Data Distribution Service 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 Data Distribution Service will maintain and deliver. 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.

The default (and most common setting) for `depth` is 1, indicating that only the most recent value should be delivered.

If `kind` is DDS::HistoryQosPolicyKind::KEEP_ALL_HISTORY_QOS, then RTI Data Distribution Service will attempt to maintain and deliver all the values of the instance to existing subscribers. The resources that RTI Data Distribution Service can use to keep this history are limited by the settings of the **RESOURCE_LIMITS** (p. 298). If the limit is reached, then the behavior of RTI Data Distribution Service will depend on the **RELIABILITY** (p. 290). If the Reliability `kind` is DDS::ReliabilityQosPolicyKind::BEST_EFFORT_RELIABILITY_QOS, then the old values will be discarded. If Reliability `kind` is RELIABLE, then RTI Data Distribution Service will block the **DDS::DataWriter** (p. 499) until it can deliver the necessary old values to all subscribers.

If `refilter` is DDS::RefilterQosPolicyKind::NONE_REFILTER_QOS, then samples written before a **DataReader** (p. 433) is matched to a **DataWriter** (p. 499) are not refiltered by the **DataWriter** (p. 499).

If `refilter` is DDS::RefilterQosPolicyKind::ALL_REFILTER_QOS, then all samples written before a **DataReader** (p. 433) is matched to a **DataWriter** (p. 499) are refiltered by the **DataWriter** (p. 499) when the **DataReader** (p. 433) is matched.

If `refilter` is DDS::RefilterQosPolicyKind::ON_DEMAND_REFILTER_QOS, then a **DataWriter** (p. 499) will only refilter samples that a **DataReader** (p. 433) requests.

6.84.3 Consistency

This QoS policy's `depth` must be consistent with the **RESOURCE_LIMITS** (p. 298) `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. 1109)

6.84.4 Member Data Documentation

6.84.4.1 HistoryQosPolicyKind DDS::HistoryQosPolicy::kind

Specifies the kind of history to be kept.

[**default**] DDS::HistoryQosPolicyKind::KEEP_LAST_HISTORY_QOS

6.84.4.2 System::Int32 DDS::HistoryQosPolicy::depth

Specifies the number of samples to be kept, when the `kind` is DDS::HistoryQosPolicyKind::KEEP_LAST_HISTORY_QOS.

If a value other than 1 (the default) is specified, it should be consistent with the settings of the **RESOURCE_LIMITS** (p. 298) policy. That is:

depth <= **DDS::ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1113)

When the `kind` is DDS::HistoryQosPolicyKind::KEEP_ALL_HISTORY_QOS, the depth has no effect. Its implied value is `infinity` (in practice limited by the settings of the **RESOURCE_LIMITS** (p. 298) policy).

[**default**] 1

[**range**] [1,100 million], <= **DDS::ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1113)

6.84.4.3 RefilterQosPolicyKind DDS::HistoryQosPolicy::refilter

<<***eXtension***>> *(p. 174)* Specifies how a writer should handle previously written samples to a new reader.

[**default**] DDS::RefilterQosPolicyKind::NONE_REFILTER_QOS

6.85 DDS::ICopyable< T > Interface Template Reference

<<eXtension>> (p. 174) *<<interface>>* (p. 175) Interface for all the user-defined data type classes that support copy.

`#include <managed_sequence.h>`

Inheritance diagram for DDS::ICopyable< T >::

6.85.1 Detailed Description

template<typename T> interface DDS::ICopyable< T >

<<eXtension>> (p. 174) *<<interface>>* (p. 175) Interface for all the user-defined data type classes that support copy.

A class implements the **DDS::ICopyable** (p. 902) interface to indicate that it allows its entire state to be replaced with the state of another object. This state copy is a deep copy, such that subsequent changes to any part of one object will not be observed in the other.

Therefore, in general, object references in this object cannot simply be reassigned to those in the source object. (Strings are an exception to this rule, because they are immutable.)

Examples:

HelloWorld.cpp.

6.86 DDS::InconsistentTopicStatus Struct Reference

DDS::StatusKind::INCONSISTENT_TOPIC_STATUS

`#include <managed_topic.h>`

Public Attributes

- System::Int32 **total_count**

  *Total cumulative count of the Topics discovered whose name matches the **DDS::Topic** (p. 1258) to which this status is attached and whose type is inconsistent with that of that **DDS::Topic** (p. 1258).*

- System::Int32 **total_count_change**

  The incremental number of inconsistent topics discovered since the last time this status was read.

6.86.1 Detailed Description

DDS::StatusKind::INCONSISTENT_TOPIC_STATUS

Entity:

DDS::Topic (p. 1258)

Listener:

DDS::TopicListener (p. 1278)

A remote **DDS::Topic** (p. 1258) will be inconsistent with the locally created **DDS::Topic** (p. 1258) if the type name of the two topics are different.

6.86.2 Member Data Documentation

6.86.2.1 System::Int32 DDS::InconsistentTopicStatus::total_count

Total cumulative count of the Topics discovered whose name matches the **DDS::Topic** (p. 1258) to which this status is attached and whose type is inconsistent with that of that **DDS::Topic** (p. 1258).

6.86.2.2 System::Int32 DDS::InconsistentTopicStatus::total_count_change

The incremental number of inconsistent topics discovered since the last time this status was read.

6.87 DDS::InstanceHandle_t Struct Reference

Type definition for an instance handle.

`#include <managed_infrastructure.h>`

Static Public Attributes

- static **InstanceHandle_t HANDLE_NIL**

  The NIL instance handle.

Properties

- bool **is_nil** `[get]`

  *Compare this handle to **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53).*

6.87.1 Detailed Description

Type definition for an instance handle.

Handle to identiy different instances of the same **DDS::Topic** (p. 1258) of a certain type.

See also:

DDS::TypedDataWriter::register_instance (p. 1370)
DDS::SampleInfo::instance_handle (p. 1153)

Examples:

HelloWorld_publisher.cpp.

6.88 DDS::InstanceHandleSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < **DDS::InstanceHandle_t** (p. 905) > .

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::InstanceHandleSeq::

6.88.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < **DDS::InstanceHandle_t** (p. 905) > .

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

DDS::InstanceHandle_t (p. 905)
DDS::Sequence (p. 1163)

6.89 DDS::InstanceStateKind Struct Reference

Indicates is the samples are from a live **DDS::DataWriter** (p. 499) or not.

`#include <managed_subscription.h>`

Properties

- static **InstanceStateKind ALIVE_INSTANCE_STATE** `[get]`

  Instance is currently in existence.

- static **InstanceStateKind NOT_ALIVE_DISPOSED_INSTANCE_STATE** `[get]`

  *Not alive disposed instance. The instance has been disposed by a **DataWriter** (p. 499).*

- static **InstanceStateKind NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** `[get]`

  *Not alive no writers for instance. None of the **DDS::DataWriter** (p. 499) objects are currently alive (according to the **LIVELINESS** (p. 286)) are writing the instance.*

- static **InstanceStateKind ANY_INSTANCE_STATE** `[get]`

  Any instance state ALIVE_INSTANCE_STATE | NOT_ALIVE_DISPOSED_INSTANCE_STATE | NOT_ALIVE_NO_WRITERS_INSTANCE_STATE.

- static **InstanceStateKind NOT_ALIVE_INSTANCE_STATE** `[get]`

  Not alive instance state NOT_ALIVE_DISPOSED_INSTANCE_STATE | NOT_ALIVE_NO_WRITERS_INSTANCE_STATE.

6.89.1 Detailed Description

Indicates is the samples are from a live **DDS::DataWriter** (p. 499) or not.

For each instance, the middleware internally maintains an instance state. The instance state can be:

- **DDS::InstanceStateKind::ALIVE_INSTANCE_STATE** (p. 908) indicates that (a) samples have been received for the instance, (b) there are live **DDS::DataWriter** (p. 499) 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::InstanceStateKind::NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 909) indicates the instance was explicitly disposed by a **DDS::DataWriter** (p. 499) by means of the dispose operation.

- **DDS::InstanceStateKind::NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 909) indicates the instance has been declared as not-alive by the **DDS::DataReader** (p. 433) because it detected that there are no live **DDS::DataWriter** (p. 499) 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. 283) is set to DDS::OwnershipQosPolicyKind::EXCLUSIVE_OWNERSHIP_QOS, then the instance state becomes **DDS::InstanceStateKind::NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 909) only if the **DDS::DataWriter** (p. 499) that "owns" the instance explicitly disposes it. The instance state becomes **DDS::InstanceStateKind::ALIVE_INSTANCE_STATE** (p. 908) again only if the **DDS::DataWriter** (p. 499) that owns the instance writes it.

- If **OWNERSHIP** (p. 283) is set to DDS::OwnershipQosPolicyKind::SHARED_OWNERSHIP_QOS, then the instance state becomes **DDS::InstanceStateKind::NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 909) if any **DDS::DataWriter** (p. 499) explicitly disposes the instance. The instance state becomes **DDS::InstanceStateKind::ALIVE_INSTANCE_STATE** (p. 908) as soon as any **DDS::DataWriter** (p. 499) writes the instance again.

The instance state available in the **DDS::SampleInfo** (p. 1148) 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.

6.89.2 Property Documentation

6.89.2.1 InstanceStateKind DDS::InstanceStateKind::ALIVE_INSTANCE_STATE [static, get]

Instance is currently in existence.

6.89.2.2 InstanceStateKind DDS::InstanceStateKind::NOT_ALIVE_DISPOSED_INSTANCE_STATE [static, get]

Not alive disposed instance. The instance has been disposed by a **DataWriter** (p. 499).

6.89.2.3 InstanceStateKind DDS::InstanceStateKind::NOT_ALIVE_NO_WRITERS_INSTANCE_STATE [static, get]

Not alive no writers for instance. None of the **DDS::DataWriter** (p. 499) objects are currently alive (according to the **LIVELINESS** (p. 286)) are writing the instance.

6.90 DDS::IntSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < System::Int32 >.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::IntSeq::

Public Member Functions

- ^ **IntSeq** ()

  Constructs an empty sequence of integers with an initial maximum of zero.

- ^ **IntSeq** (System::Int32 max)

  Constructs an empty sequence of integers with the given initial maximum.

- ^ **IntSeq** (**IntSeq**^ ints)

  Constructs a new sequence containing the given integers.

6.90.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < System::Int32 >.

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

System::Int32
DDS::Sequence (p. 1163)

6.90.2 Constructor & Destructor Documentation

6.90.2.1 DDS::IntSeq::IntSeq () `[inline]`

Constructs an empty sequence of integers with an initial maximum of zero.

6.90.2.2 DDS::IntSeq::IntSeq (System::Int32 *max*) `[inline]`

Constructs an empty sequence of integers with the given initial maximum.

6.90.2.3 DDS::IntSeq::IntSeq (IntSeq^ *ints*) [inline]

Constructs a new sequence containing the given integers.

Parameters:

ints the initial contents of this sequence

6.91 DDS::ITopicDescription Interface Reference

<<*interface*>> *(p. 175)* Base class for **DDS::Topic** (p. 1258), **DDS::ContentFilteredTopic** (p. 419), and **DDS::MultiTopic** (p. 984).

`#include <managed_topic.h>`

Inheritance diagram for DDS::ITopicDescription::

Public Member Functions

^ System::String^ **get_type_name** ()

Get the associated `type_name`.

^ System::String^ **get_name** ()

Get the name used to create this DDS::TopicDescription .

^ **DomainParticipant**^ **get_participant** ()

Get the ***DDS::DomainParticipant*** *(p. 577) to which the DDS::TopicDescription belongs.*

6.91.1 Detailed Description

<<*interface*>> *(p. 175)* Base class for **DDS::Topic** (p. 1258), **DDS::ContentFilteredTopic** (p. 419), and **DDS::MultiTopic** (p. 984).

DDS::TopicDescription 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. 1385).

DDS::TopicDescription has also a `name` that allows it to be retrieved locally.

See also:

DDS::TypeSupport (p. 1385), **FooTypeSupport** (p. 884)

6.91.2 Member Function Documentation

6.91.2.1 System::String ^ DDS::ITopicDescription::get_type_name ()

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. 885) method.

Thus, the `type_name` implies an association with a corresponding **DDS::TypeSupport** (p. 1385) and this DDS::TopicDescription.

Returns:

the type name. The returned type name is valid until the DDS::TopicDescription is deleted.

Postcondition:

The result is non-NULL.

See also:

DDS::TypeSupport (p. 1385), **FooTypeSupport** (p. 884)

Implemented in **DDS::ContentFilteredTopic** (p. 424), **DDS::MultiTopic** (p. 987), and **DDS::Topic** (p. 1263).

6.91.2.2 System::String ^ DDS::ITopicDescription::get_name ()

Get the name used to create this DDS::TopicDescription .

Returns:

the name used to create this DDS::TopicDescription. The returned topic name is valid until the DDS::TopicDescription is deleted.

Postcondition:

The result is non-NULL.

Implemented in **DDS::ContentFilteredTopic** (p. 424), **DDS::MultiTopic** (p. 988), and **DDS::Topic** (p. 1264).

6.91.2.3 DomainParticipant ^ DDS::ITopicDescription::get_participant ()

Get the **DDS::DomainParticipant** (p. 577) to which the DDS::TopicDescription belongs.

Returns:

The **DDS::DomainParticipant** (p. 577) to which the DDS::TopicDescription belongs.

Postcondition:

The result is non-NULL.

Implemented in **DDS::ContentFilteredTopic** (p. 425), **DDS::MultiTopic** (p. 988), and **DDS::Topic** (p. 1264).

6.92 DDS::KeyedBytes Struct Reference

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

`#include <managed_keyedbytes.h>`

Inheritance diagram for DDS::KeyedBytes::

Public Member Functions

- ^ **KeyedBytes** ()

  Default Constructor.

- ^ **KeyedBytes** (System::Int32 size)

  Constructor that specifies the allocated sizes.

- ^ virtual System::Boolean **copy_from** (**KeyedBytes**^ src)

  Copy src into this object.

Public Attributes

- ^ System::String^ **key**

  Instance key associated with the specified value.

- ^ System::Int32 **length**

  Number of bytes to serialize.

- ^ System::Int32 **offset**

  Offset from which to start serializing bytes .

- ^ array< System::Byte >^ **value**

  DDS::Bytes *(p. 388) array value.*

6.92.1 Detailed Description

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

6.92.2 Constructor & Destructor Documentation

6.92.2.1 DDS::KeyedBytes::KeyedBytes ()

Default Constructor.

The default constructor initializes the newly created object with empty key, null value, zero length, and zero offset.

6.92.2.2 DDS::KeyedBytes::KeyedBytes (System::Int32 *size*)

Constructor that specifies the allocated sizes.

After this method is called, key is initialized with the empty string and length and offset are set to zero.

Parameters:

size *<<in>>* *(*p. 175*)* Size of the allocated bytes array.

6.92.3 Member Function Documentation

6.92.3.1 virtual System::Boolean DDS::KeyedBytes::copy_from (KeyedBytes^ *src*) [virtual]

Copy src into this object.

This method performs a deep copy of `src` and it allocates memory for the value if required.

Parameters:

src *<<in>>* *(*p. 175*)* Object to copy from.

Returns:

true if success. Otherwise, false.

Exceptions:

ArgumentNullException if `src` is null.

6.92.4 Member Data Documentation

6.92.4.1 System::String ^ DDS::KeyedBytes::key

Instance key associated with the specified value.

6.92.4.2 System::Int32 DDS::KeyedBytes::length

Number of bytes to serialize.

6.92.4.3 System::Int32 DDS::KeyedBytes::offset

Offset from which to start serializing bytes .

The first position of the bytes array has offset 0.

6.92.4.4 array<System::Byte> ^ DDS::KeyedBytes::value

DDS::Bytes (p. 388) array value.

6.93 DDS::KeyedBytesDataReader Class Reference

<<*interface*>> *(p. 175)* Instantiates `DataReader` (p. 433) < **DDS::KeyedBytes** (p. 915) >.

`#include <managed_keyedbytesSupport.h>`

Inheritance diagram for DDS::KeyedBytesDataReader::

Public Member Functions

- ^ System::String^ **get_key_value** (**DDS::InstanceHandle_t**% handle)
  <<**eXtension**>> (p. 174) *Retrieve the instance* `key` *that corresponds to an instance* `handle`.

- ^ **InstanceHandle_t lookup_instance** (System::String^ key)
  <<**eXtension**>> (p. 174) *Retrieve the instance* `handle` *that corresponds to an instance* `key`.

6.93.1 Detailed Description

<<*interface*>> *(p. 175)* Instantiates `DataReader` (p. 433) < **DDS::KeyedBytes** (p. 915) >.

See also:

DDS::TypedDataReader (p. 1338)
DDS::DataReader (p. 433)

6.93.2 Member Function Documentation

6.93.2.1 System::String ^ DDS::KeyedBytesDataReader::get_key_value (DDS::InstanceHandle_t% *handle*) [inline]

<<*eXtension*>> *(p. 174)* Retrieve the instance `key` that corresponds to an instance `handle`.

See also:

DDS::TypedDataReader::get_key_value (p. 1365)

6.93.2.2 InstanceHandle_t DDS::KeyedBytesDataReader::lookup_instance (System::String^ *key*) [inline]

<<*eXtension*>> *(p. 174)* Retrieve the instance `handle` that corresponds to an instance `key`.

See also:

DDS::TypedDataReader::lookup_instance (p. 1366)

6.94 DDS::KeyedBytesDataWriter Class Reference

<<*interface*>> *(p. 175)* Instantiates `DataWriter` (p. 499) < **DDS::KeyedBytes** (p. 915) >.

`#include <managed_keyedbytesSupport.h>`

Inheritance diagram for DDS::KeyedBytesDataWriter::

Public Member Functions

- ^ **InstanceHandle_t register_instance** (System::String^ key)

  <<**eXtension**>> (p. 174) *Informs RTI Data Distribution Service that the application will be modifying a particular instance.*

- ^ **InstanceHandle_t register_instance_w_timestamp** (System::String^ key, **DDS::Time_t**% source_timestamp)

  <<**eXtension**>> (p. 174) *Performs the same functions as* ***DDS::KeyedBytesDataWriter::register_instance*** *(p. 922) except that the application provides the value for the* `source_timestamp`.

- ^ void **unregister_instance** (System::String^ key, **DDS::InstanceHandle_t**% handle)

  <<**eXtension**>> (p. 174) *Reverses the action of* ***DDS::KeyedBytesDataWriter::register_instance*** *(p. 922).*

- ^ void **unregister_instance_w_timestamp** (System::String^ key, **DDS::InstanceHandle_t**% handle, **DDS::Time_t**% source_timestamp)

  <<**eXtension**>> (p. 174) *Performs the same function as* ***DDS::KeyedBytesDataWriter::unregister_instance*** *(p. 922) except that it also provides the value for the* `source_timestamp`.

- ^ void **write** (System::String^ key, array< System::Byte >^octets, System::Int32 offset, System::Int32 length, **DDS::InstanceHandle_t**% handle)

  <<**eXtension**>> (p. 174) *Modifies the value of a* ***DDS::KeyedBytes*** *(p. 915) data instance.*

- ^ void **write** (System::String^ key, **ByteSeq**^ octets, **DDS::InstanceHandle_t**% handle)

<<**eXtension**>> (p. 174) *Modifies the value of a* ***DDS::KeyedBytes*** *(p. 915) data instance.*

^ void **write_w_timestamp** (System::String^ key, array< System::Byte >^octets, System::Int32 offset, System::Int32 length, **DDS::InstanceHandle_t**% handle, **DDS::Time_t**% source_timestamp)

Performs the same function as ***DDS::KeyedBytesDataWriter::write*** *(p. 923) except that it also provides the value for the* `source_timestamp`.

^ void **write_w_timestamp** (System::String^ key, **ByteSeq**^ octets, **DDS::InstanceHandle_t**% handle, **DDS::Time_t**% source_timestamp)

Performs the same function as ***DDS::KeyedBytesDataWriter::write*** *(p. 923) except that it also provides the value for the* `source_timestamp`.

^ void **dispose** (System::String^ key, **DDS::InstanceHandle_t**% instance_handle)

<<**eXtension**>> (p. 174) *Requests the middleware to delete the data.*

^ void **dispose_w_timestamp** (System::String^ key, **DDS::InstanceHandle_t**% instance_handle, **DDS::Time_t**% source_timestamp)

<<**eXtension**>> (p. 174) *Performs the same functions as* ***DDS::KeyedBytesDataWriter::dispose*** *(p. 925) except that the application provides the value for the* `source_timestamp` *that is made available to* ***DDS::DataReader*** *(p. 433) objects by means of the* `source_timestamp` *attribute inside the* ***DDS::SampleInfo*** *(p. 1148).*

^ System::String^ **get_key_value** (**DDS::InstanceHandle_t**% handle)

<<**eXtension**>> (p. 174) *Retrieve the instance* `key` *that corresponds to an instance* `handle`.

^ **InstanceHandle_t lookup_instance** (System::String^ key)

<<**eXtension**>> (p. 174) *Retrieve the instance* `handle` *that corresponds to an instance* `key`.

6.94.1 Detailed Description

<<***interface***>> *(p. 175)* Instantiates `DataWriter` (p. 499) < **DDS::KeyedBytes** (p. 915) >.

See also:

DDS::TypedDataWriter (p. 1368)

DDS::DataWriter (p. 499)

6.94.2 Member Function Documentation

6.94.2.1 InstanceHandle_t DDS::KeyedBytesDataWriter::register_instance (System::String^ *key*)

<<*eXtension*>> *(p. 174)* Informs RTI Data Distribution Service that the application will be modifying a particular instance.

See also:

DDS::TypedDataWriter::register_instance (p. 1370)

6.94.2.2 InstanceHandle_t DDS::KeyedBytesDataWriter::register_instance_w_timestamp (System::String^ *key*, DDS::Time_t% *source_timestamp*)

<<*eXtension*>> *(p. 174)* Performs the same functions as **DDS::KeyedBytesDataWriter::register_instance** (p. 922) except that the application provides the value for the `source_timestamp`.

See also:

DDS::TypedDataWriter::register_instance_w_timestamp (p. 1371)

6.94.2.3 void DDS::KeyedBytesDataWriter::unregister_instance (System::String^ *key*, DDS::InstanceHandle_t% *handle*)

<<*eXtension*>> *(p. 174)* Reverses the action of **DDS::KeyedBytesDataWriter::register_instance** (p. 922).

See also:

DDS::TypedDataWriter::unregister_instance (p. 1372)

6.94.2.4 void DDS::KeyedBytesDataWriter::unregister_instance_w_timestamp (System::String^ *key*, DDS::InstanceHandle_t% *handle*, DDS::Time_t% *source_timestamp*)

<<*eXtension*>> *(p. 174)* Performs the same function as **DDS::KeyedBytesDataWriter::unregister_instance** (p. 922) except that it also provides the value for the `source_timestamp`.

See also:

DDS::TypedDataWriter::unregister_instance_w_timestamp (p. 1374)

6.94.2.5 void DDS::KeyedBytesDataWriter::write (System::String^ *key*, array< System::Byte >^ *octets*, System::Int32 *offset*, System::Int32 *length*, DDS::InstanceHandle_t% *handle*)

*<<**eXtension**>>* *(p. 174)* Modifies the value of a **DDS::KeyedBytes** (p. 915) data instance.

Parameters:

key *<<**in**>>* *(p. 175)* Instance key.

octets *<<**in**>>* *(p. 175)* Array of bytes to be published.

offset *<<**in**>>* *(p. 175)* Offset from which to start publishing.

length *<<**in**>>* *(p. 175)* Number of bytes to be published.

handle *<<**in**>>* *(p. 175)* Either the handle returned by a previous call to DDS::KeyedOctetsDataWriter::register_instance, or else the special value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53). See **DDS::TypedDataWriter::write** (p. 1376).

See also:

DDS::TypedDataWriter::write (p. 1376)

6.94.2.6 void DDS::KeyedBytesDataWriter::write (System::String^ *key*, ByteSeq^ *octets*, DDS::InstanceHandle_t% *handle*)

*<<**eXtension**>>* *(p. 174)* Modifies the value of a **DDS::KeyedBytes** (p. 915) data instance.

Parameters:

key *<<**in**>>* *(p. 175)* Instance key.

octets *<<**in**>>* *(p. 175)* **Sequence** (p. 1163) of bytes to be published.

handle *<<**in**>>* *(p. 175)* Either the handle returned by a previous call to DDS::KeyedOctetsDataWriter::register_instance, or else the special value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53). See **DDS::TypedDataWriter::write** (p. 1376).

See also:

DDS::TypedDataWriter::write (p. 1376)

6.94.2.7 void DDS::KeyedBytesDataWriter::write_w_timestamp (System::String^ *key*, array< System::Byte >^ *octets*, System::Int32 *offset*, System::Int32 *length*, DDS::InstanceHandle_t% *handle*, DDS::Time_t% *source_timestamp*)

Performs the same function as **DDS::KeyedBytesDataWriter::write** (p. 923) except that it also provides the value for the `source_timestamp`.

Parameters:

key *<<in>>* *(p. 175)* Instance key.

octets *<<in>>* *(p. 175)* Array of bytes to be published.

offset *<<in>>* *(p. 175)* Offset from which to start publishing.

length *<<in>>* *(p. 175)* Number of bytes to be published.

handle *<<in>>* *(p. 175)* Either the handle returned by a previous call to DDS::KeyedOctetsDataWriter::register_instance, or else the special value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53). See **DDS::TypedDataWriter::write** (p. 1376).

source_timestamp *<<in>>* *(p. 175)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation. See **DDS::TypedDataWriter::write_w_timestamp** (p. 1378). Cannot be NULL.

See also:

DDS::TypedDataWriter::write (p. 1376)

6.94.2.8 void DDS::KeyedBytesDataWriter::write_w_timestamp (System::String^ *key*, ByteSeq^ *octets*, DDS::InstanceHandle_t% *handle*, DDS::Time_t% *source_timestamp*)

Performs the same function as **DDS::KeyedBytesDataWriter::write** (p. 923) except that it also provides the value for the `source_timestamp`.

Parameters:

key *<<in>>* *(p. 175)* Instance key.

octets *<<in>>* *(p. 175)* **Sequence** (p. 1163) of bytes to be published.

handle *<<in>>* *(p. 175)* Either the handle returned by a previous call to DDS::KeyedOctetsDataWriter::register_instance, or else the special value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53). See **DDS::TypedDataWriter::write** (p. 1376).

source_timestamp *<<**in**>>* *(p. 175)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation. See **DDS::TypedDataWriter::write_w_timestamp** (p. 1378). Cannot be NULL.

See also:

DDS::TypedDataWriter::write (p. 1376)

6.94.2.9 void DDS::KeyedBytesDataWriter::dispose (System::String^ *key*, DDS::InstanceHandle_t% *instance_handle*)

*<<**eXtension**>>* *(p. 174)* Requests the middleware to delete the data.

See also:

DDS::TypedDataWriter::dispose (p. 1379)

6.94.2.10 void DDS::KeyedBytesDataWriter::dispose_w_timestamp (System::String^ *key*, DDS::InstanceHandle_t% *instance_handle*, DDS::Time_t% *source_timestamp*)

*<<**eXtension**>>* *(p. 174)* Performs the same functions as **DDS::KeyedBytesDataWriter::dispose** (p. 925) except that the application provides the value for the `source_timestamp` that is made available to **DDS::DataReader** (p. 433) objects by means of the `source_timestamp` attribute inside the **DDS::SampleInfo** (p. 1148).

See also:

DDS::TypedDataWriter::dispose_w_timestamp (p. 1381)

6.94.2.11 System::String ^ DDS::KeyedBytesDataWriter::get_key_value (DDS::InstanceHandle_t% *handle*)

*<<**eXtension**>>* *(p. 174)* Retrieve the instance `key` that corresponds to an instance `handle`.

See also:

DDS::TypedDataWriter::get_key_value (p. 1383)

6.94.2.12 InstanceHandle_t DDS::KeyedBytesDataWriter::lookup_instance (System::String^ *key*)

<<*eXtension*>> *(*p. *174)* Retrieve the instance `handle` that corresponds to an instance `key`.

See also:

DDS::TypedDataWriter::lookup_instance (p. 1384)

6.95 DDS::KeyedBytesSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < **DDS::KeyedBytes** (p. 915) >.

`#include <managed_keyedbytes.h>`

Public Member Functions

- ^ **KeyedBytesSeq** ()

  Constructs an empty sequence of ***DDS::KeyedBytes*** *(p. 915) objects with an initial maximum of zero.*

- ^ **KeyedBytesSeq** (System::Int32 initialMaximum)

  Constructs an empty sequence of ***DDS::KeyedBytes*** *(p. 915) objects with the given initial maximum.*

- ^ **KeyedBytesSeq** (**KeyedBytesSeq**^ src)

  Copy constructor.

6.95.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < **DDS::KeyedBytes** (p. 915) >.

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

DDS::KeyedBytes (p. 915)

6.95.2 Constructor & Destructor Documentation

6.95.2.1 DDS::KeyedBytesSeq::KeyedBytesSeq () `[inline]`

Constructs an empty sequence of **DDS::KeyedBytes** (p. 915) objects with an initial maximum of zero.

6.95.2.2 DDS::KeyedBytesSeq::KeyedBytesSeq (System::Int32 *initialMaximum*) `[inline]`

Constructs an empty sequence of **DDS::KeyedBytes** (p. 915) objects with the given initial maximum.

6.95.2.3 DDS::KeyedBytesSeq::KeyedBytesSeq (KeyedBytesSeq^ *src*) [inline]

Copy constructor.

6.96 DDS::KeyedBytesTypeSupport Class Reference

<<*interface*>> *(p. 175)* **DDS::KeyedBytes** (p. 915) type support.

`#include <managed_keyedbytesSupport.h>`

Inherits DDS::TypedTypeSupport< T >.

Static Public Member Functions

- ^ static System::String^ **get_type_name** ()

  Get the default name for the ***DDS::KeyedBytes*** *(p. 915) type.*

- ^ static void **print_data** (**KeyedBytes**^ a_data)

  <<**eXtension**>> (p. 174) *Print value of data type to standard out.*

- ^ static void **register_type** (**DDS::DomainParticipant**^ participant, System::String^ type_name)

  Allows an application to communicate to RTI Data Distribution Service the existence of the ***DDS::KeyedBytes*** *(p. 915) data type.*

- ^ static void **unregister_type** (**DDS::DomainParticipant**^ participant, System::String^ type_name)

  Allows an application to unregister the ***DDS::KeyedBytes*** *(p. 915) data type from RTI Data Distribution Service. After calling unregister_type, no further communication using this type is possible.*

6.96.1 Detailed Description

<<*interface*>> *(p. 175)* **DDS::KeyedBytes** (p. 915) type support.

6.96.2 Member Function Documentation

6.96.2.1 static System::String ^ DDS::KeyedBytesTypeSupport::get_type_name () [static]

Get the default name for the **DDS::KeyedBytes** (p. 915) type.

Can be used for calling **DDS::KeyedBytesTypeSupport::register_type** (p. 930) or creating **DDS::Topic** (p. 1258).

Returns:

default name for the **DDS::KeyedBytes** (p. 915) type.

See also:

DDS::KeyedBytesTypeSupport::register_type (p. 930)
DDS::DomainParticipant::create_topic (p. 621)

6.96.2.2 static void DDS::KeyedBytesTypeSupport::print_data (KeyedBytes^ *a_data*) [static]

<<eXtension>> *(p. 174)* Print value of data type to standard out.

The *generated* implementation of the operation knows how to print value of a data type.

Parameters:

a_data *<<in>>* *(p. 175)* **DDS::KeyedBytes** (p. 915) to be printed.

6.96.2.3 static void DDS::KeyedBytesTypeSupport::register_type (DDS::DomainParticipant^ *participant*, System::String^ *type_name*) [static]

Allows an application to communicate to RTI Data Distribution Service the existence of the **DDS::KeyedBytes** (p. 915) data type.

By default, The **DDS::KeyedBytes** (p. 915) built-in type is automatically registered when a **DomainParticipant** (p. 577) is created using the type_name returned by **DDS::KeyedBytesTypeSupport::get_type_name** (p. 929). Therefore, the usage of this function is optional and it is only required when the automatic built-in type registration is disabled using the participant property "dds.builtin_type.auto_register".

This method can also be used to register the same **DDS::KeyedBytesTypeSupport** (p. 929) with a **DDS::DomainParticipant** (p. 577) using different values for the type_name.

If `register_type` is called multiple times with the same **DDS::DomainParticipant** (p. 577) and `type_name`, the second (and subsequent) registrations are ignored by the operation.

Parameters:

participant *<<in>>* *(p. 175)* the **DDS::DomainParticipant** (p. 577) to register the data type **DDS::Bytes** (p. 388) with. Cannot be null.

type_name *<<in>> (p. 175)* the type name under with the data type **DDS::KeyedBytes** (p. 915) is registered with the participant; this type name is used when creating a new **DDS::Topic** (p. 1258). (See **DDS::DomainParticipant::create_topic** (p. 621).) The name may not be null or longer than 255 characters.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_PreconditionNotMet** (p. 1123) or **DDS::Retcode_OutOfResources** (p. 1122).

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. 621)

6.96.2.4 static void DDS::KeyedBytesTypeSupport::unregister_type (DDS::DomainParticipant^ *participant*, System::String^ *type_name*) [static]

Allows an application to unregister the **DDS::KeyedBytes** (p. 915) data type from RTI Data Distribution Service. After calling unregister_type, no further communication using this type is possible.

Precondition:

The **DDS::KeyedBytes** (p. 915) type with `type_name` is registered with the participant and all **DDS::Topic** (p. 1258) objects referencing the type have been destroyed. If the type is not registered with the participant, or if any **DDS::Topic** (p. 1258) is associated with the type, the operation will fail with **DDS::Retcode_Error** (p. 1116).

Postcondition:

All information about the type is removed from RTI Data Distribution Service. No further communication using this type is possible.

Parameters:

participant *<<in>> (p. 175)* the **DDS::DomainParticipant** (p. 577) to unregister the data type **DDS::KeyedBytes** (p. 915) from. Cannot be null.

type_name *<<in>> (p. 175)* the type name under with the data type **DDS::KeyedBytes** (p. 915) is registered with the participant. The name should match a name that has been previously used to register a type with the participant. Cannot be null.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_BadParameter** (p. 1115) or **DDS::Retcode_Error** (p. 1116)

MT Safety:

SAFE.

See also:

DDS::KeyedBytesTypeSupport::register_type (p. 930)

6.97 DDS::KeyedString Struct Reference

Keyed string built-in type.

`#include <managed_keyedstring.h>`

Inheritance diagram for DDS::KeyedString::

Public Member Functions

- ^ **KeyedString** ()

  Default Constructor.

- ^ virtual System::Boolean **copy_from** (**KeyedString**^ src)

  Copy src into this object.

Public Attributes

- ^ System::String^ **key**

  Instance key associated with the specified value.

- ^ System::String^ **value**

  String value.

6.97.1 Detailed Description

Keyed string built-in type.

6.97.2 Constructor & Destructor Documentation

6.97.2.1 DDS::KeyedString::KeyedString ()

Default Constructor.

The default constructor initializes the newly created object with empty key and value.

6.97.3 Member Function Documentation

6.97.3.1 virtual System::Boolean DDS::KeyedString::copy_from (KeyedString^ *src*) [virtual]

Copy src into this object.

This method performs a deep copy of `src`.

Parameters:

src *<<in>>* *(p. 175)* Object to copy from.

Returns:

true if success. Otherwise, false.

Exceptions:

ArgumentNullException if `src` is null.

6.97.4 Member Data Documentation

6.97.4.1 System::String ^ DDS::KeyedString::key

Instance key associated with the specified value.

6.97.4.2 System::String ^ DDS::KeyedString::value

String value.

6.98 DDS::KeyedStringDataReader Class Reference

<<*interface*>> *(p. 175)* Instantiates `DataReader` (p. 433) < **DDS::KeyedString** (p. 933) >.

`#include <managed_keyedstringSupport.h>`

Inheritance diagram for DDS::KeyedStringDataReader::

Public Member Functions

- ^ System::String^ **get_key_value** (**DDS::InstanceHandle_t**% handle)
  <<**eXtension**>> (p. 174) *Retrieve the instance* `key` *that corresponds to an instance* `handle`.

- ^ **InstanceHandle_t lookup_instance** (System::String^ key)
  <<**eXtension**>> (p. 174) *Retrieve the instance* `handle` *that corresponds to an instance* `key`.

6.98.1 Detailed Description

<<*interface*>> *(p. 175)* Instantiates `DataReader` (p. 433) < **DDS::KeyedString** (p. 933) >.

See also:

DDS::TypedDataReader (p. 1338)
DDS::DataReader (p. 433)

6.98.2 Member Function Documentation

6.98.2.1 System::String ^ DDS::KeyedStringDataReader::get_key_value (DDS::InstanceHandle_t% *handle*) [inline]

<<*eXtension*>> *(p. 174)* Retrieve the instance `key` that corresponds to an instance `handle`.

See also:

DDS::TypedDataReader::get_key_value (p. 1365)

6.98.2.2 InstanceHandle_t DDS::KeyedStringDataReader::lookup_instance (System::String^ *key*) [inline]

<<*eXtension*>> *(*p. *174**)* Retrieve the instance `handle` that corresponds to an instance `key`.

See also:

DDS::TypedDataReader::lookup_instance (p. 1366)

6.99 DDS::KeyedStringDataWriter Class Reference

<<*interface*>> *(p. 175)* Instantiates `DataWriter` (p. 499) < **DDS::KeyedString** (p. 933) >.

`#include <managed_keyedstringSupport.h>`

Inheritance diagram for DDS::KeyedStringDataWriter::

Public Member Functions

- ^ **InstanceHandle_t register_instance** (System::String^ key)

  <<**eXtension**>> (p. 174) *Informs RTI Data Distribution Service that the application will be modifying a particular instance.*

- ^ **InstanceHandle_t register_instance_w_timestamp** (System::String^ key, **DDS::Time_t**% source_timestamp)

  <<**eXtension**>> (p. 174) *Performs the same functions as* ***DDS::KeyedStringDataWriter::register_instance*** *(p. 938) except that the application provides the value for the* `source_timestamp`.

- ^ void **unregister_instance** (System::String^ key, **DDS::InstanceHandle_t**% handle)

  <<**eXtension**>> (p. 174) *Reverses the action of* ***DDS::KeyedStringDataWriter::register_instance*** *(p. 938).*

- ^ void **unregister_instance_w_timestamp** (System::String^ key, **DDS::InstanceHandle_t**% handle, **DDS::Time_t**% source_timestamp)

  <<**eXtension**>> (p. 174) *Performs the same function as* ***DDS::KeyedStringDataWriter::unregister_instance*** *(p. 939) except that it also provides the value for the* `source_timestamp`.

- ^ void **write** (System::String^ key, System::String^ str, **DDS::InstanceHandle_t**% handle)

  <<**eXtension**>> (p. 174) *Modifies the value of a* ***DDS::KeyedString*** *(p. 933) data instance.*

- ^ void **write_w_timestamp** (System::String^ key, System::String^ str, **DDS::InstanceHandle_t**% handle, **DDS::Time_t**% source_timestamp)

<<**eXtension**>> (p. 174) *Performs the same function as* ***DDS::KeyedStringDataWriter::write*** *(p. 940) except that it also provides the value for the* `source_timestamp`.

^ void **dispose** (System::String^ key, **DDS::InstanceHandle_t**% instance_handle)

<<**eXtension**>> (p. 174) *Requests the middleware to delete the data.*

^ void **dispose_w_timestamp** (System::String^ key, **DDS::InstanceHandle_t**% instance_handle, **DDS::Time_t**% source_timestamp)

<<**eXtension**>> (p. 174) *Performs the same functions as* ***DDS::KeyedStringDataWriter::dispose*** *(p. 940) except that the application provides the value for the* `source_timestamp` *that is made available to* ***DDS::DataReader*** *(p. 433) objects by means of the* `source_timestamp` *attribute inside the* ***DDS::SampleInfo*** *(p. 1148).*

^ System::String^ **get_key_value** (**DDS::InstanceHandle_t**% handle)

<<**eXtension**>> (p. 174) *Retrieve the instance* `key` *that corresponds to an instance* `handle`.

^ **InstanceHandle_t lookup_instance** (System::String^ key)

<<**eXtension**>> (p. 174) *Retrieve the instance* `handle` *that corresponds to an instance* `key`.

6.99.1 Detailed Description

<<***interface***>> *(p. 175)* Instantiates `DataWriter` (p. 499) < **DDS::KeyedString** (p. 933) >.

See also:

DDS::TypedDataWriter (p. 1368)
DDS::DataWriter (p. 499)

6.99.2 Member Function Documentation

6.99.2.1 InstanceHandle_t DDS::KeyedStringDataWriter::register_instance (System::String^ *key*)

<<***eXtension***>> *(p. 174)* Informs RTI Data Distribution Service that the application will be modifying a particular instance.

See also:

DDS::TypedDataWriter::register_instance (p. 1370)

6.99.2.2 InstanceHandle_t DDS::KeyedStringDataWriter::register_instance_w_timestamp (System::String^ *key*, DDS::Time_t% *source_timestamp*)

<<eXtension>> *(p. 174)* Performs the same functions as **DDS::KeyedStringDataWriter::register_instance** (p. 938) except that the application provides the value for the `source_timestamp`.

See also:

DDS::TypedDataWriter::register_instance_w_timestamp (p. 1371)

6.99.2.3 void DDS::KeyedStringDataWriter::unregister_instance (System::String^ *key*, DDS::InstanceHandle_t% *handle*)

<<eXtension>> *(p. 174)* Reverses the action of **DDS::KeyedStringDataWriter::register_instance** (p. 938).

See also:

DDS::TypedDataWriter::unregister_instance (p. 1372)

6.99.2.4 void DDS::KeyedStringDataWriter::unregister_instance_w_timestamp (System::String^ *key*, DDS::InstanceHandle_t% *handle*, DDS::Time_t% *source_timestamp*)

<<eXtension>> *(p. 174)* Performs the same function as **DDS::KeyedStringDataWriter::unregister_instance** (p. 939) except that it also provides the value for the `source_timestamp`.

See also:

DDS::TypedDataWriter::unregister_instance_w_timestamp (p. 1374)

6.99.2.5 void DDS::KeyedStringDataWriter::write (System::String^ *key*, System::String^ *str*, DDS::InstanceHandle_t% *handle*)

<<*eXtension*>> *(p. 174)* Modifies the value of a **DDS::KeyedString** (p. 933) data instance.

See also:

DDS::TypedDataWriter::write (p. 1376)

6.99.2.6 void DDS::KeyedStringDataWriter::write_w_timestamp (System::String^ *key*, System::String^ *str*, DDS::InstanceHandle_t% *handle*, DDS::Time_t% *source_timestamp*)

<<*eXtension*>> *(p. 174)* Performs the same function as **DDS::KeyedStringDataWriter::write** (p. 940) except that it also provides the value for the `source_timestamp`.

See also:

DDS::TypedDataWriter::write_w_timestamp (p. 1378)

6.99.2.7 void DDS::KeyedStringDataWriter::dispose (System::String^ *key*, DDS::InstanceHandle_t% *instance_handle*)

<<*eXtension*>> *(p. 174)* Requests the middleware to delete the data.

See also:

DDS::TypedDataWriter::dispose (p. 1379)

6.99.2.8 void DDS::KeyedStringDataWriter::dispose_w_timestamp (System::String^ *key*, DDS::InstanceHandle_t% *instance_handle*, DDS::Time_t% *source_timestamp*)

<<*eXtension*>> *(p. 174)* Performs the same functions as **DDS::KeyedStringDataWriter::dispose** (p. 940) except that the application provides the value for the `source_timestamp` that is made available to **DDS::DataReader** (p. 433) objects by means of the `source_timestamp` attribute inside the **DDS::SampleInfo** (p. 1148).

See also:

DDS::TypedDataWriter::dispose_w_timestamp (p. 1381)

6.99.2.9 System::String ^ DDS::KeyedStringDataWriter::get_key_value (DDS::InstanceHandle_t% *handle*)

<<***eXtension***>> *(p. 174)* Retrieve the instance `key` that corresponds to an instance `handle`.

See also:

DDS::TypedDataWriter::get_key_value (p. 1383)

6.99.2.10 InstanceHandle_t DDS::KeyedStringDataWriter::lookup_instance (System::String^ *key*)

<<***eXtension***>> *(p. 174)* Retrieve the instance `handle` that corresponds to an instance `key`.

See also:

DDS::TypedDataWriter::lookup_instance (p. 1384)

6.100 DDS::KeyedStringSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < **DDS::KeyedString** (p. 933) > .

`#include <managed_keyedstring.h>`

Public Member Functions

- ^ **KeyedStringSeq** ()

  Constructs an empty sequence of ***DDS::KeyedString*** (p. 933) *objects with an initial maximum of zero.*

- ^ **KeyedStringSeq** (System::Int32 initialMaximum)

  Constructs an empty sequence of ***DDS::KeyedString*** (p. 933) *objects with the given initial maximum.*

- ^ **KeyedStringSeq** (**KeyedStringSeq**^ src)

  Copy constructor.

6.100.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < **DDS::KeyedString** (p. 933) > .

Instantiates:

<<***generic***>> (p. 175) **DDS::Sequence** (p. 1163)

See also:

DDS::KeyedString (p. 933)

6.100.2 Constructor & Destructor Documentation

6.100.2.1 DDS::KeyedStringSeq::KeyedStringSeq () `[inline]`

Constructs an empty sequence of **DDS::KeyedString** (p. 933) objects with an initial maximum of zero.

6.100.2.2 DDS::KeyedStringSeq::KeyedStringSeq (System::Int32 *initialMaximum*) `[inline]`

Constructs an empty sequence of **DDS::KeyedString** (p. 933) objects with the given initial maximum.

6.100.2.3 DDS::KeyedStringSeq::KeyedStringSeq (KeyedStringSeq^ *src*) [inline]

Copy constructor.

6.101 DDS::KeyedStringTypeSupport Class Reference

<<***interface***>> *(p. 175)* Keyed string type support.

`#include <managed_keyedstringSupport.h>`

Inherits DDS::TypedTypeSupport< T >.

Static Public Member Functions

- ^ static System::String^ **get_type_name** ()

  Get the default name for the ***DDS::KeyedString*** *(p. 933) type.*

- ^ static void **print_data** (**KeyedString**^ a_data)

  <<**eXtension**>> (p. 174) *Print value of data type to standard out.*

- ^ static void **register_type** (**DDS::DomainParticipant**^ participant, System::String^ type_name)

  Allows an application to communicate to RTI Data Distribution Service the existence of the ***DDS::KeyedString*** *(p. 933) data type.*

- ^ static void **unregister_type** (**DDS::DomainParticipant**^ participant, System::String^ type_name)

  Allows an application to unregister the ***DDS::KeyedString*** *(p. 933) data type from RTI Data Distribution Service. After calling unregister_type, no further communication using this type is possible.*

6.101.1 Detailed Description

<<***interface***>> *(p. 175)* Keyed string type support.

6.101.2 Member Function Documentation

6.101.2.1 static System::String ^ DDS::KeyedStringTypeSupport::get_type_name () [static]

Get the default name for the **DDS::KeyedString** (p. 933) type.

Can be used for calling **DDS::KeyedStringTypeSupport::register_type** (p. 945) or creating **DDS::Topic** (p. 1258).

Returns:

default name for the **DDS::KeyedString** (p. 933) type.

See also:

DDS::KeyedStringTypeSupport::register_type (p. 945)
DDS::DomainParticipant::create_topic (p. 621)

6.101.2.2 static void DDS::KeyedStringTypeSupport::print_data (KeyedString^ *a_data*) [static]

<<eXtension>> *(p. 174)* Print value of data type to standard out.

The *generated* implementation of the operation knows how to print value of a data type.

Parameters:

a_data *<<in>>* *(p. 175)* **DDS::KeyedString** (p. 933) to be printed.

6.101.2.3 static void DDS::KeyedStringTypeSupport::register_type (DDS::DomainParticipant^ *participant*, System::String^ *type_name*) [static]

Allows an application to communicate to RTI Data Distribution Service the existence of the **DDS::KeyedString** (p. 933) data type.

By default, The **DDS::KeyedString** (p. 933) built-in type is automatically registered when a **DomainParticipant** (p. 577) is created using the type_name returned by **DDS::KeyedStringTypeSupport::get_type_name** (p. 944). Therefore, the usage of this function is optional and it is only required when the automatic built-in type registration is disabled using the participant property "dds.builtin_type.auto_register".

This method can also be used to register the same **DDS::KeyedStringTypeSupport** (p. 944) with a **DDS::DomainParticipant** (p. 577) using different values for the type_name.

If `register_type` is called multiple times with the same **DDS::DomainParticipant** (p. 577) and `type_name`, the second (and subsequent) registrations are ignored by the operation.

Parameters:

participant *<<in>>* *(p. 175)* the **DDS::DomainParticipant** (p. 577) to register the data type **DDS::KeyedString** (p. 933) with. Cannot be null.

type_name *<<in>> (p. 175)* the type name under with the data type **DDS::KeyedString** (p. 933) is registered with the participant; this type name is used when creating a new **DDS::Topic** (p. 1258). (See **DDS::DomainParticipant::create_topic** (p. 621).) The name may not be null or longer than 255 characters.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_PreconditionNotMet** (p. 1123) or **DDS::Retcode_OutOfResources** (p. 1122).

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. 621)

6.101.2.4 static void DDS::KeyedStringTypeSupport::unregister_type (DDS::DomainParticipant^ *participant*, System::String^ *type_name*) [static]

Allows an application to unregister the **DDS::KeyedString** (p. 933) data type from RTI Data Distribution Service. After calling unregister_type, no further communication using this type is possible.

Precondition:

The **DDS::KeyedString** (p. 933) type with `type_name` is registered with the participant and all **DDS::Topic** (p. 1258) objects referencing the type have been destroyed. If the type is not registered with the participant, or if any **DDS::Topic** (p. 1258) is associated with the type, the operation will fail with **DDS::Retcode_Error** (p. 1116).

Postcondition:

All information about the type is removed from RTI Data Distribution Service. No further communication using this type is possible.

Parameters:

participant *<<in>> (p. 175)* the **DDS::DomainParticipant** (p. 577) to unregister the data type **DDS::KeyedString** (p. 933) from. Cannot be null.

type_name *<<in>>* *(p. 175)* the type name under with the data type **DDS::KeyedString** (p. 933) is registered with the participant. The name should match a name that has been previously used to register a type with the participant. Cannot be null.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_BadParameter** (p. 1115) or **DDS::Retcode_Error** (p. 1116)

MT Safety:

SAFE.

See also:

DDS::KeyedStringTypeSupport::register_type (p. 945)

6.102 DDS::LatencyBudgetQosPolicy Struct Reference

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.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_latencybudget_qos_policy_name** ()

  *Stringified human-readable name for **DDS::LatencyBudgetQosPolicy** (p. 948).*

Public Attributes

- **Duration_t duration**

  Duration of the maximum acceptable delay.

6.102.1 Detailed Description

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. 1258), **DDS::DataReader** (p. 433), **DDS::DataWriter** (p. 499)

Status:

DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS,
DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS

Properties:

RxO (p. 268) = YES
Changeable (p. 269) = **YES** (p. 269)

See also:

DDS::PublishModeQosPolicy (p. 1077)
DDS::FlowController (p. 867)

6.102.2 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 Data Distribution Service can optimize its internal operation.

RTI Data Distribution Service uses it in conjunction with DDS::PublishModeQosPolicyKind::ASYNCHRONOUS_PUBLISH_MODE_QOS **DDS::DataWriter** (p. 499) instances associated with a DDS::FlowControllerSchedulingPolicy::EDF_FLOW_CONTROLLER_SCHED_POLICY **DDS::FlowController** (p. 867) only. Together with the time of write, **DDS::LatencyBudgetQosPolicy::duration** (p. 949) determines the deadline of each individual sample. RTI Data Distribution Service uses this information to prioritize the sending of asynchronously published data; see **DDS::AsynchronousPublisherQosPolicy** (p. 371).

6.102.3 Compatibility

The value offered is considered compatible with the value requested if and only if the inequality *offered duration* <= *requested duration* evaluates to 'TRUE'.

6.102.4 Member Data Documentation

6.102.4.1 Duration_t DDS::LatencyBudgetQosPolicy::duration

Duration of the maximum acceptable delay.

[**default**] 0 (meaning minimize the delay)

6.103 DDS::LifespanQosPolicy Struct Reference

Specifies how long the data written by the **DDS::DataWriter** (p. 499) is considered valid.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_lifespan_qos_policy_name** ()

  *Stringified human-readable name for **DDS::LifespanQosPolicy** (p. 950).*

Public Attributes

- **Duration_t duration**

  Maximum duration for the data's validity.

6.103.1 Detailed Description

Specifies how long the data written by the **DDS::DataWriter** (p. 499) is considered valid.

Each data sample written by the **DDS::DataWriter** (p. 499) 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. 433) caches as well as from the transient and persistent information caches.

The expiration time of each sample from the **DDS::DataWriter** (p. 499)'s cache is computed by adding the duration specified by this QoS policy to the sample's source timestamp. The expiration time of each sample from the **DDS::DataReader** (p. 433)'s cache is computed by adding the duration to the reception timestamp.

See also:

DDS::TypedDataWriter::write (p. 1376)
DDS::TypedDataWriter::write_w_timestamp (p. 1378)

Entity:

DDS::Topic (p. 1258), **DDS::DataWriter** (p. 499)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **YES** (p. 269)

6.103.2 Usage

The Lifespan QoS policy can be used to control how much data is stored by RTI Data Distribution Service. Even if it is configured to store "all" of the data sent or received for a topic (see **DDS::HistoryQosPolicy** (p. 898)), 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. 1153)
DDS::SampleInfo::reception_timestamp (p. 1156)

6.103.3 Member Data Documentation

6.103.3.1 Duration_t DDS::LifespanQosPolicy::duration

Maximum duration for the data's validity.

[**default**] **DDS::Duration_t::DURATION_INFINITE** (p. 253)

[**range**] [1 nanosec, 1 year] or **DDS::Duration_t::DURATION_INFINITE** (p. 253)

6.104 DDS::Listener Class Reference

<<*interface*>> *(p. 175)* Abstract base class for all **Listener** (p. 952) interfaces.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::Listener::

6.104.1 Detailed Description

<<*interface*>> *(p. 175)* Abstract base class for all **Listener** (p. 952) interfaces.

Entity:

DDS::Entity (p. 845)

QoS:

QoS Policies (p. 260)

Status:

Status Kinds (p. 238)

All the supported kinds of concrete **DDS::Listener** (p. 952) interfaces (one per concrete **DDS::Entity** (p. 845) type) derive from this root and add methods whose prototype depends on the concrete **Listener** (p. 952).

Listeners provide a way for RTI Data Distribution Service 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. 952) instance may be shared among multiple entities if you so desire. Consequently, the provided parameter contains a reference to the concerned **DDS::Entity** (p. 845).

6.104.2 Access to Plain Communication Status

The general mapping between the plain communication statuses (see **Status Kinds** (p. 238)) 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. 238).

- `on_<communication_status>` is available on the relevant **DDS::Entity** (p. 845) as well as those that embed it, as expressed in the following figure:

- When the application attaches a listener on an entity, it must set a mask. The mask indicates to RTI Data Distribution Service which operations are enabled within the listener (cf. operation **DDS::Entity** (p. 845) set_listener()).

- When a plain communication status changes, RTI Data Distribution Service 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::DomainParticipant** (p. 577)) and to set dedicated behaviors only where needed.

6.104.3 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 Data Distribution Service tries to trigger the **DDS::SubscriberListener::on_data_on_readers** (p. 1229) with a parameter of the related **DDS::Subscriber** (p. 1201);

- If this does not succeed (there is no listener or the operation is not enabled), RTI Data Distribution Service tries to trigger **DDS::DataReaderListener::on_data_available** (p. 463) on all the related **DDS::DataReaderListener** (p. 461) objects, with a parameter of the related **DDS::DataReader** (p. 433).

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. 433) objects by calling **DDS::Subscriber::get_datareaders** (p. 1218) on the related **DDS::Subscriber** (p. 1201) and then get the data by calling **DDS::TypedDataReader::read** (p. 1341) or **DDS::TypedDataReader::take** (p. 1342) on the returned **DDS::DataReader** (p. 433) objects), or it wants to treat each **DDS::DataReader** (p. 433) independently and it may choose the second option (and then get the data by calling **DDS::TypedDataReader::read** (p. 1341) or **DDS::TypedDataReader::take** (p. 1342) on the related **DDS::DataReader** (p. 433)).

Note that if **DDS::SubscriberListener::on_data_on_readers** (p. 1229) is called, RTI Data Distribution Service will *not* try to call **DDS::DataReaderListener::on_data_available** (p. 463). However, an application can force a call to the **DDS::DataReader** (p. 433) objects that have data by calling **DDS::Subscriber::notify_datareaders** (p. 1219).

6.104.4 Operations Allowed in Listener Callbacks

The operations that are allowed in **DDS::Listener** (p. 952) callbacks depend on the **DDS::ExclusiveAreaQosPolicy** (p. 862) QoS policy of the **DDS::Entity** (p. 845) to which the **DDS::Listener** (p. 952) is attached – or in the case of a **DDS::DataWriter** (p. 499) of **DDS::DataReader** (p. 433) listener, on the **DDS::ExclusiveAreaQosPolicy** (p. 862) QoS of the parent **DDS::Publisher** (p. 1044) or **DDS::Subscriber** (p. 1201). For instance, the **DDS::ExclusiveAreaQosPolicy** (p. 862) settings of a **DDS::Subscriber** (p. 1201) will determine which operations are allowed within the callbacks of the listeners associated with all the DataReaders created through that **DDS::Subscriber** (p. 1201).

Note: these restrictions do not apply to builtin topic listener callbacks.

Regardless of whether **DDS::ExclusiveAreaQosPolicy::use_shared_exclusive_area** (p. 864) is set to true or false, the following operations are *not* allowed:

- Within any listener callback, deleting the entity to which the **DDS::Listener** (p. 952) is attached
- Within a **DDS::Topic** (p. 1258) listener callback, any operations on any subscribers, readers, publishers or writers

An attempt to call a disallowed method from within a callback will result in **DDS::Retcode_IllegalOperation** (p. 1117).

If **DDS::ExclusiveAreaQosPolicy::use_shared_exclusive_area** (p. 864) is set to false, the setting which allows more concurrency among RTI Data Distribution Service threads, the following are *not* allowed:

- Within any listener callback, creating any entity
- Within any listener callback, deleting any entity
- Within any listener callback, enabling any entity
- Within any listener callback, setting the QoS of any entities
- Within a **DDS::DataReader** (p. 433) or **DDS::Subscriber** (p. 1201) listener callback, invoking any operation on any other **DDS::Subscriber** (p. 1201) or on any **DDS::DataReader** (p. 433) belonging to another **DDS::Subscriber** (p. 1201).
- Within a **DDS::DataReader** (p. 433) or **DDS::Subscriber** (p. 1201) listener callback, invoking any operation on any **DDS::Publisher** (p. 1044) (or on any **DDS::DataWriter** (p. 499) belonging to such a **DDS::Publisher** (p. 1044)) that has **DDS::ExclusiveAreaQosPolicy::use_shared_exclusive_area** (p. 864) set to true.
- Within a **DDS::DataWriter** (p. 499) of **DDS::Publisher** (p. 1044) listener callback, invoking any operation on another **Publisher** (p. 1044) or on a **DDS::DataWriter** (p. 499) belonging to another **DDS::Publisher** (p. 1044).
- Within a **DDS::DataWriter** (p. 499) of **DDS::Publisher** (p. 1044) listener callback, invoking any operation on any **DDS::Subscriber** (p. 1201) or **DDS::DataReader** (p. 433).

An attempt to call a disallowed method from within a callback will result in **DDS::Retcode_IllegalOperation** (p. 1117).

The above limitations can be lifted by setting **DDS::ExclusiveAreaQosPolicy::use_shared_exclusive_area** (p. 864) to true on the **DDS::Publisher** (p. 1044) or **DDS::Subscriber** (p. 1201) (or on the **DDS::Publisher** (p. 1044)/ **DDS::Subscriber** (p. 1201) of the **DDS::DataWriter** (p. 499)/**DDS::DataReader** (p. 433)) to which the listener is attached. However, the application will pay the cost of reduced concurrency between the affected publishers and subscribers.

See also:

EXCLUSIVE_AREA (p. 352)
Status Kinds (p. 238)
DDS::WaitSet (p. 1411), **DDS::Condition** (p. 408)

6.105 DDS::LivelinessChangedStatus Struct Reference

DDS::StatusKind::LIVELINESS_CHANGED_STATUS

`#include <managed_subscription.h>`

Public Attributes

- System::Int32 **alive_count**

  The total count of currently alive ***DDS::DataWriter*** *(p. 499) entities that write the* ***DDS::Topic*** *(p. 1258) the* ***DDS::DataReader*** *(p. 433) reads.*

- System::Int32 **not_alive_count**

  The total count of currently not_alive ***DDS::DataWriter*** *(p. 499) entities that write the* ***DDS::Topic*** *(p. 1258) the* ***DDS::DataReader*** *(p. 433) reads.*

- System::Int32 **alive_count_change**

  The change in the alive_count since the last time the listener was called or the status was read.

- System::Int32 **not_alive_count_change**

  The change in the not_alive_count since the last time the listener was called or the status was read.

- **InstanceHandle_t last_publication_handle**

  An instance handle to the last remote writer to change its liveliness.

6.105.1 Detailed Description

DDS::StatusKind::LIVELINESS_CHANGED_STATUS

Examples:

HelloWorld_subscriber.cpp.

6.105.2 Member Data Documentation

6.105.2.1 System::Int32 DDS::LivelinessChangedStatus::alive_count

The total count of currently alive **DDS::DataWriter** (p. 499) entities that write the **DDS::Topic** (p. 1258) the **DDS::DataReader** (p. 433) reads.

6.105.2.2 System::Int32 DDS::LivelinessChangedStatus::not_alive_count

The total count of currently not_alive **DDS::DataWriter** (p. 499) entities that write the **DDS::Topic** (p. 1258) the **DDS::DataReader** (p. 433) reads.

6.105.2.3 System::Int32 DDS::LivelinessChangedStatus::alive_count_change

The change in the alive_count since the last time the listener was called or the status was read.

6.105.2.4 System::Int32 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.

6.105.2.5 InstanceHandle_t DDS::LivelinessChangedStatus::last_publication_handle

An instance handle to the last remote writer to change its liveliness.

6.106 DDS::LivelinessLostStatus Struct Reference

DDS::StatusKind::LIVELINESS_LOST_STATUS

`#include <managed_publication.h>`

Public Attributes

- System::Int32 **total_count**

  *Total cumulative number of times that a previously-alive **DDS::DataWriter** (p. 499) became not alive due to a failure to to actively signal its liveliness within the offered liveliness period.*

- System::Int32 **total_count_change**

  The incremental changees in total_count since the last time the listener was called or the status was read.

6.106.1 Detailed Description

DDS::StatusKind::LIVELINESS_LOST_STATUS

Entity:

DDS::DataWriter (p. 499)

Listener:

DDS::DataWriterListener (p. 524)

The liveliness that the **DDS::DataWriter** (p. 499) has committed through its **DDS::LivelinessQosPolicy** (p. 960) was not respected; thus **DDS::DataReader** (p. 433) entities will consider the **DDS::DataWriter** (p. 499) as no longer "alive/active".

6.106.2 Member Data Documentation

6.106.2.1 System::Int32 DDS::LivelinessLostStatus::total_count

Total cumulative number of times that a previously-alive **DDS::DataWriter** (p. 499) became not alive due to a failure to to actively signal its liveliness within the offered liveliness period.

This count does not change when an already not alive **DDS::DataWriter** (p. 499) simply remains not alive for another liveliness period.

6.106.2.2 System::Int32 DDS::LivelinessLostStatus::total_count_change

The incremental changees in total_count since the last time the listener was called or the status was read.

6.107 DDS::LivelinessQosPolicy Struct Reference

Specifies and configures the mechanism that allows **DDS::DataReader** (p. 433) entities to detect when **DDS::DataWriter** (p. 499) entities become disconnected or "dead.".

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_livelines_qos_policy_name** ()

  *Stringified human-readable name for **DDS::LivelinessQosPolicy** (p. 960).*

Public Attributes

- **LivelinessQosPolicyKind kind**

  The kind of liveliness desired.

- **Duration_t lease_duration**

  *The duration within which a **DDS::Entity** (p. 845) must be asserted, or else it is assumed to be not alive.*

6.107.1 Detailed Description

Specifies and configures the mechanism that allows **DDS::DataReader** (p. 433) entities to detect when **DDS::DataWriter** (p. 499) entities become disconnected or "dead.".

Liveliness must be asserted at least once every `lease_duration` otherwise RTI Data Distribution Service will assume the corresponding **DDS::Entity** (p. 845) or is no longer alive.

The liveliness status of a **DDS::Entity** (p. 845) is used to maintain instance ownership in combination with the setting of the **OWNERSHIP** (p. 283) policy. The application is also informed via **DDS::Listener** (p. 952) when an **DDS::Entity** (p. 845) is no longer alive.

A **DDS::DataReader** (p. 433) requests that liveliness of writers is maintained by the requested means and loss of liveliness is detected with delay not to exceed the `lease_duration`.

A **DDS::DataWriter** (p. 499) commits to signalling its liveliness using the stated means at intervals not to exceed the `lease_duration`.

Listeners are used to notify a **DDS::DataReader** (p. 433) of loss of liveliness and **DDS::DataWriter** (p. 499) 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. 499) 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 run time 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. 1258), **DDS::DataReader** (p. 433), **DDS::DataWriter** (p. 499)

Status:

DDS::StatusKind::LIVELINESS_LOST_STATUS,
DDS::LivelinessLostStatus (p. 958);
DDS::StatusKind::LIVELINESS_CHANGED_STATUS,
DDS::LivelinessChangedStatus (p. 956);
DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS,
DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS

Properties:

RxO (p. 268) = YES
Changeable (p. 269) = **UNTIL ENABLE** (p. 269)

6.107.2 Usage

This policy controls the mechanism and parameters used by RTI Data Distribution Service to ensure that particular entities on the network are still alive. The liveliness can also affect the ownership of a particular instance, as determined by the **OWNERSHIP** (p. 283) 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::LivelinessQosPolicyKind::AUTOMATIC_LIVELINESS_QOS 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 Data Distribution Service takes responsibility for renewing the leases at the required rates and thus, as long as the local process where a **DDS::DomainParticipant** (p. 577) is running and the link connecting it to remote participants remains connected, the entities within the **DDS::DomainParticipant** (p. 577) will be considered alive. This requires the lowest overhead.

The manual settings (DDS::LivelinessQosPolicyKind::MANUAL_BY_PARTICIPANT_LIVELINESS_QOS, DDS::LivelinessQosPolicyKind::MANUAL_BY_TOPIC_LIVELINESS_QOS) require the application on the publishing side to periodically assert the liveliness before the lease expires to indicate the corresponding **DDS::Entity** (p. 845) is still alive. The action can be explicit by calling the **DDS::DataWriter::assert_liveliness** (p. 508) 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::LivelinessQosPolicyKind::MANUAL_BY_PARTICIPANT_LIVELINESS_QOS requires only that one **DDS::Entity** (p. 845) within a participant is asserted to be alive to deduce all other **DDS::Entity** (p. 845) objects within the same **DDS::DomainParticipant** (p. 577) are also alive.

- The setting DDS::LivelinessQosPolicyKind::MANUAL_BY_TOPIC_LIVELINESS_QOS requires that at least one instance within the **DDS::DataWriter** (p. 499) is asserted.

Changes in **LIVELINESS** (p. 286) must be detected by the Service with a time-granularity greater or equal to the `lease_duration`. This ensures that the value of the **DDS::LivelinessChangedStatus** (p. 956) is updated at least once during each `lease_duration` and the related Listeners and **DDS::WaitSet** (p. 1411) s are notified within a `lease_duration` from the time the **LIVELINESS** (p. 286) changed.

6.107.3 Compatibility

The value offered is considered compatible with the value requested if and only if the following conditions are met:

- the inequality *offered kind* >= *requested kind* evaluates to 'TRUE'. For the purposes of this inequality, the values of DDS::LivelinessQosPolicyKind kind are considered ordered such that: DDS::LivelinessQosPolicyKind::AUTOMATIC_LIVELINESS_QOS

< DDS::LivelinessQosPolicyKind::MANUAL_BY_PARTICIPANT_LIVELINESS_QOS < DDS::LivelinessQosPolicyKind::MANUAL_BY_TOPIC_LIVELINESS_QOS.

ˆ the inequality *offered* `lease_duration` <= *requested* `lease_duration` evaluates to true.

See also:

RELATIONSHIP BETWEEN REGISTRATION, LIVELINESS and OWNERSHIP (p. 995)

6.107.4 Member Data Documentation

6.107.4.1 LivelinessQosPolicyKind DDS::LivelinessQosPolicy::kind

The kind of liveliness desired.

[**default**] DDS::LivelinessQosPolicyKind::AUTOMATIC_LIVELINESS_QOS

6.107.4.2 Duration_t DDS::LivelinessQosPolicy::lease_duration

The duration within which a **DDS::Entity** (p. 845) must be asserted, or else it is assumed to be not alive.

[**default**] **DDS::Duration_t::DURATION_INFINITE** (p. 253)

[**range**] [0,1 year] or **DDS::Duration_t::DURATION_INFINITE** (p. 253)

6.108 DDS::LoanableSequence< E > Class Template Reference

A sequence implementation used internally by the middleware to efficiently manage memory during **DDS::TypedDataReader::read** (p. 1341) and **DDS::TypedDataReader::take** (p. 1342) operations.

`#include <managed_sequence.h>`

Inheritance diagram for DDS::LoanableSequence< E >::

Public Member Functions

- virtual E **get_at** (System::Int32 i) overridesealed

  Get the `i-th` *element for a* `const` *sequence.*

- virtual System::Boolean **copy_from_no_alloc** (**Sequence**< E >^src) overridesealed

  Copy elements from another sequence, only if the destination sequence has enough capacity.

- virtual void **unloan** () overridesealed

  Return the loaned buffer in the sequence and set the maximum to 0.

Properties

- virtual System::Int32 **maximum** `[set]`

  The current maximum number of elements that can be stored in this sequence.

6.108.1 Detailed Description

template<typename E> class DDS::LoanableSequence< E >

A sequence implementation used internally by the middleware to efficiently manage memory during **DDS::TypedDataReader::read** (p. 1341) and **DDS::TypedDataReader::take** (p. 1342) operations.

Applications are not expected to use this type directly.

6.108.2 Member Function Documentation

6.108.2.1 template<typename E> virtual E DDS::LoanableSequence< E >::get_at (System::Int32 *i*) [override, sealed, virtual]

Get the `i-th` element for a `const` sequence.

Parameters:

i index of element to access, must be >= 0 and less than **DDS::Sequence::length** (p. 1171)

Returns:

the `i-th` element

Reimplemented from **DDS::Sequence< E >** (p. 1166).

6.108.2.2 template<typename E> virtual System::Boolean DDS::LoanableSequence< E >::copy_from_no_alloc (Sequence< E >^ *src_seq*) [override, sealed, virtual]

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 == true

Postcondition:

this::length == src_seq::length
this[i] == src_seq[i] for 0 <= i < target_seq::length
this::owned == true

Parameters:

src_seq *<<in>>* *(p. 175)* the sequence from which to copy

Returns:

true if the sequence was successfully copied; false otherwise.

Note:

If the pre-conditions are not met, the operator will print a message to stdout and leave this sequence unchanged.

See also:

DDS::Sequence::copy_from_no_alloc (p. 1170)(DDS::Sequence<T>^)

Reimplemented from **DDS::Sequence< E >** (p. 1170).

6.108.2.3 template<typename E> virtual void DDS::LoanableSequence< E >::unloan () [override, sealed, virtual]

Return the loaned buffer in the sequence and set the maximum to 0.

This method affects only the state of this sequence; it does not change the contents of the buffer in any way.

Only the user who originally loaned a buffer should return that loan, as the user may have dependencies on that memory known only to them. Unloaning someone else's buffer may cause unspecified problems. For example, suppose a sequence is loaning memory from a custom memory pool. A user of the sequence likely has no way to release the memory back into the pool, so unloaning the sequence buffer would result in a resource leak. If the user were to then re-loan a different buffer, the original creator of the sequence would have no way to discover, when freeing the sequence, that the loan no longer referred to its own memory and would thus not free the user's memory properly, exacerbating the situation and leading to undefined behavior.

Precondition:

owned == false

Postcondition:

owned == true
maximum == 0

Returns:

true if the preconditions were met. Otherwise false. The function only fails if the pre-conditions are not met, in which case it leaves the sequence unmodified.

See also:

DDS::Sequence<T>::loan (p. 1167)(array<T>^, System::Int32), DDS::Sequence::loan_discontiguous, **DDS::Sequence::maximum** (p. 1172)

Reimplemented from **DDS::Sequence< E >** (p. 1168).

6.108.3 Property Documentation

6.108.3.1 template<typename E> virtual System:: Int32 DDS::LoanableSequence< E >::maximum [set]

The current maximum number of elements that can be stored in this sequence.

Getting the property:

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 **DDS::Sequence::maximum** (p. 1172) operation.

See also:

DDS::Sequence::length (p. 1171)

Setting the property:

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 == true

Postcondition:

owned == true
length == MINIMUM(original length, new_max)

Parameters:

new_max Must be >= 0.

Reimplemented from **DDS::Sequence< E >** (p. 1172).

6.109 DDS::Locator_t Class Reference

<<***eXtension***>> *(p. 174)* Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports.

`#include <managed_infrastructure.h>`

Public Attributes

- ^ System::Int32 **kind**

  The kind of locator.

- ^ System::UInt32 **port**

  the port number

- ^ array< System::Byte >^ **address**

  A ***DDS::Locator_t::LOCATOR_ADDRESS_LENGTH_MAX*** *(p. 46) octet field to hold the IP address.*

Properties

- ^ static System::Int32 **LOCATOR_ADDRESS_LENGTH_MAX** `[get]`

  Declares length of address field in locator.

- ^ static **Locator_t**^ **LOCATOR_INVALID** `[get]`

  An invalid locator.

- ^ static System::Int32 **LOCATOR_KIND_INVALID** `[get]`

  Locator of this kind is invalid.

- ^ static System::UInt32 **LOCATOR_PORT_INVALID** `[get]`

  An invalid port.

- ^ static array< System::Byte >^ **LOCATOR_ADDRESS_INVALID** `[get]`

  An invalid address.

- ^ static System::Int32 **LOCATOR_KIND_UDPv4** `[get]`

  A locator for a UDPv4 address.

^ static System::Int32 **LOCATOR_KIND_UDPv6** [get]

A locator for a UDPv6 address.

^ static System::Int32 **LOCATOR_KIND_RESERVED** [get]

Locator of this kind is reserved.

^ static System::Int32 **LOCATOR_KIND_SHMEM** [get]

A locator for an address acessed via shared memory.

6.109.1 Detailed Description

<<***eXtension***>> *(p. 174)* Type used to represent the addressing information needed to send a message to an RTPS Endpoint using one of the supported transports.

6.109.2 Member Data Documentation

6.109.2.1 System::Int32 DDS::Locator_t::kind

The kind of locator.

If the **Locator_t** (p. 968) kind is **DDS::Locator_t::LOCATOR_KIND_UDPv4** (p. 46), the address contains an IPv4 address. In this case, the leading 12 octets of the **DDS::Locator_t::address** (p. 969) must be zero. The last 4 octets of **DDS::Locator_t::address** (p. 969) are used to store the IPv4 address.

If the **Locator_t** (p. 968) kind is **DDS::Locator_t::LOCATOR_KIND_UDPv6** (p. 46), 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. 969) field.

6.109.2.2 System::UInt32 DDS::Locator_t::port

the port number

6.109.2.3 array<System::Byte> ^ DDS::Locator_t::address

A **DDS::Locator_t::LOCATOR_ADDRESS_LENGTH_MAX** (p. 46) octet field to hold the IP address.

6.110 DDS::LocatorFilter_t Class Reference

The QoS policy used to report the configuration of a MultiChannel **DataWriter** (p. 499) as part of **DDS::PublicationBuiltinTopicData** (p. 1030).

`#include <managed_infrastructure.h>`

Public Attributes

- ^ **LocatorSeq**^ **locators**

  Sequence *(p. 1163) containing from one to four* ***DDS::Locator_t*** *(p. 968), used to specify the multicast address locators of an individual channel within a MultiChannel* ***DataWriter*** *(p. 499).*

- ^ System::String^ **filter_expression**

  A logical expression used to determine the data that will be published in the channel.

6.110.1 Detailed Description

The QoS policy used to report the configuration of a MultiChannel **DataWriter** (p. 499) as part of **DDS::PublicationBuiltinTopicData** (p. 1030).

Entity:

DDS::PublicationBuiltinTopicData (p. 1030)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)
Specifies the configuration of an individual channel within a MultiChannel **DataWriter** (p. 499).

QoS:

DDS::LocatorFilterQosPolicy (p. 972)

6.110.2 Member Data Documentation

6.110.2.1 LocatorSeq ^ DDS::LocatorFilter_t::locators

Sequence (p. 1163) containing from one to four **DDS::Locator_t** (p. 968), used to specify the multicast address locators of an individual channel within a MultiChannel **DataWriter** (p. 499).

[**default**] Empty sequence.

6.110.2.2 System::String ^ DDS::LocatorFilter_t::filter_expression

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. 973)

See also:

Queries and Filters Syntax (p. 184)

[**default**] NULL (invalid value)

6.111 DDS::LocatorFilterQosPolicy Class Reference

The QoS policy used to report the configuration of a MultiChannel **DataWriter** (p. 499) as part of **DDS::PublicationBuiltinTopicData** (p. 1030).

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_locator_filter_qos_policy_name** ()

  *Stringified human-readable name for **DDS::LocatorFilterQosPolicy** (p. 972).*

Public Attributes

- **LocatorFilterSeq**^ **locator_filters**

  *A sequence of **DDS::LocatorFilter_t** (p. 970). Each **DDS::LocatorFilter_t** (p. 970) reports the configuration of a single channel of a MultiChannel **DataWriter** (p. 499).*

- System::String^ **filter_name**

  *Name of the filter class used to describe the filter expressions of a MultiChannel **DataWriter** (p. 499).*

6.111.1 Detailed Description

The QoS policy used to report the configuration of a MultiChannel **DataWriter** (p. 499) as part of **DDS::PublicationBuiltinTopicData** (p. 1030).

Entity:

DDS::PublicationBuiltinTopicData (p. 1030)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)

6.111.2 Member Data Documentation

6.111.2.1 LocatorFilterSeq ^ DDS::LocatorFilterQosPolicy::locator_filters

A sequence of **DDS::LocatorFilter_t** (p. 970). Each **DDS::LocatorFilter_t** (p. 970) reports the configuration of a single channel of a MultiChannel **DataWriter** (p. 499).

A sequence length of zero indicates the **DDS::MultiChannelQosPolicy** (p. 981) is not in use.

[**default**] Empty sequence.

6.111.2.2 System::String ^ DDS::LocatorFilterQosPolicy::filter_name

Name of the filter class used to describe the filter expressions of a MultiChannel **DataWriter** (p. 499).

The following builtin filters are supported: **DDS::DomainParticipant::SQLFILTER_NAME** (p. 40) and **DDS::DomainParticipant::STRINGMATCHFILTER_NAME** (p. 41).

[**default**] **DDS::DomainParticipant::STRINGMATCHFILTER_NAME** (p. 41)

6.112 DDS::LocatorFilterSeq Class Reference

Declares IDL `sequence<` **DDS::LocatorFilter_t** (p. 970) `>`.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::LocatorFilterSeq::

6.112.1 Detailed Description

Declares IDL `sequence<` **DDS::LocatorFilter_t** (p. 970) `>`.

A sequence of **DDS::LocatorFilter_t** (p. 970) used to report the channels' properties. If the length of the sequence is zero, the **DDS::MultiChannelQosPolicy** (p. 981) is not in use.

Instantiates:

*<<**generic**>>* *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

DDS::LocatorFilter_t (p. 970)

6.113 DDS::LocatorSeq Class Reference

Declares IDL `sequence` < **DDS::Locator_t** (p. 968) >.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::LocatorSeq::

6.113.1 Detailed Description

Declares IDL `sequence` < **DDS::Locator_t** (p. 968) >.

See also:

DDS::Locator_t (p. 968)

6.114 DDS::LongDouble Struct Reference

Defines an extra-precision floating-point data type, equivalent to IDL/CDR `long double`.

`#include <managed_infrastructure.h>`

6.114.1 Detailed Description

Defines an extra-precision floating-point data type, equivalent to IDL/CDR `long double`.

A 128-bit floating-point value.

6.115 DDS::LongDoubleSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < **DDS::LongDouble** (p. 976) >.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::LongDoubleSeq::

Public Member Functions

- ^ **LongDoubleSeq** ()

  Constructs an empty sequence of long doubles with an initial maximum of zero.

- ^ **LongDoubleSeq** (System::Int32 max)

  Constructs an empty sequence of long doubles with the given initial maximum.

- ^ **LongDoubleSeq** (**LongDoubleSeq**^ doubles)

  Constructs a new sequence containing the given long doubles.

6.115.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < **DDS::LongDouble** (p. 976) >.

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

DDS::LongDouble (p. 976)
DDS::Sequence (p. 1163)

6.115.2 Constructor & Destructor Documentation

6.115.2.1 DDS::LongDoubleSeq::LongDoubleSeq () `[inline]`

Constructs an empty sequence of long doubles with an initial maximum of zero.

6.115.2.2 DDS::LongDoubleSeq::LongDoubleSeq (System::Int32 *max*) [inline]

Constructs an empty sequence of long doubles with the given initial maximum.

6.115.2.3 DDS::LongDoubleSeq::LongDoubleSeq (LongDoubleSeq^ *doubles*) [inline]

Constructs a new sequence containing the given long doubles.

Parameters:

doubles the initial contents of this sequence

6.116 DDS::LongSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < System::Int64 >.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::LongSeq::

Public Member Functions

- ^ **LongSeq** ()

  Constructs an empty sequence of long integers with an initial maximum of zero.

- ^ **LongSeq** (System::Int32 max)

  Constructs an empty sequence of long integers with the given initial maximum.

- ^ **LongSeq** (**LongSeq**^ longs)

  Constructs a new sequence containing the given longs.

6.116.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < System::Int64 >.

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

System::Int64
DDS::Sequence (p. 1163)

6.116.2 Constructor & Destructor Documentation

6.116.2.1 DDS::LongSeq::LongSeq () `[inline]`

Constructs an empty sequence of long integers with an initial maximum of zero.

6.116.2.2 DDS::LongSeq::LongSeq (System::Int32 *max*) [inline]

Constructs an empty sequence of long integers with the given initial maximum.

6.116.2.3 DDS::LongSeq::LongSeq (LongSeq^ *longs*) [inline]

Constructs a new sequence containing the given longs.

Parameters:

longs the initial contents of this sequence

6.117 DDS::MultiChannelQosPolicy Class Reference

Configures the ability of a **DataWriter** (p. 499) to send data on different multicast groups (addresses) based on the value of the data.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- ^ static System::String^ **get_multichannel_qos_policy_name** ()

  *Stringified human-readable name for **DDS::MultiChannelQosPolicy** (p. 981).*

Public Attributes

- ^ **ChannelSettingsSeq^ channels**

  *A sequence of **DDS::ChannelSettings_t** (p. 403) used to configure the channels' properties. If the length of the sequence is zero, the QoS policy will be ignored.*

- ^ System::String^ **filter_name**

  *Name of the filter class used to describe the filter expressions of a MultiChannel **DataWriter** (p. 499).*

6.117.1 Detailed Description

Configures the ability of a **DataWriter** (p. 499) 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. 499) across multiple channels. A *channel* is defined by a filter expression and a sequence of multicast locators.

Entity:

DDS::DataWriter (p. 499)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)

6.117.2 Usage

By using this QoS, a **DDS::DataWriter** (p. 499) 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 applications code calls **DDS::TypedDataWriter::write** (p. 1376), 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** (p. 1258)). 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.

6.117.3 Member Data Documentation

6.117.3.1 ChannelSettingsSeq ^ DDS::MultiChannelQosPolicy::channels

A sequence of **DDS::ChannelSettings_t** (p. 403) 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. 981) is not in use.

The sequence length cannot be greater than **DDS::DomainParticipantResourceLimitsQosPolicy::channel_seq_max_length** (p. 704).

[**default**] Empty sequence.

6.117.3.2 System::String ^ DDS::MultiChannelQosPolicy::filter_name

Name of the filter class used to describe the filter expressions of a MultiChannel **DataWriter** (p. 499).

The following builtin filters are supported: **DDS::DomainParticipant::SQLFILTER_NAME** (p. 40) and **DDS::DomainParticipant::STRINGMATCHFILTER_NAME** (p. 41).

[**default**] **DDS::DomainParticipant::STRINGMATCHFILTER_NAME** (p. 41)

6.118 DDS::MultiTopic Class Reference

[**Not supported (optional)**] <<***interface***>> *(*p. 175*)* A specialization of DDS::TopicDescription that allows subscriptions that combine/filter/rearrange data coming from several topics.

`#include <managed_topic.h>`

Inheritance diagram for DDS::MultiTopic::

Public Member Functions

- ^ System::String^ **get_subscription_expression** ()

  Get the expression for this ***DDS::MultiTopic*** *(*p. 984*).*

- ^ void **get_expression_parameters** (**StringSeq**^ parameters)

  Get the expression parameters.

- ^ void **set_expression_parameters** (**StringSeq**^ parameters)

  Set the `expression_parameters`*.*

- ^ virtual System::String^ **get_type_name** ()

  Get the associated `type_name`*.*

- ^ virtual System::String^ **get_name** ()

  Get the name used to create this DDS::TopicDescription .

- ^ virtual **DomainParticipant**^ **get_participant** ()

  Get the ***DDS::DomainParticipant*** *(*p. 577*) to which the DDS::TopicDescription belongs.*

Static Public Member Functions

- ^ static **MultiTopic**^ **narrow** (**ITopicDescription**^ topic_description)

  Narrow the given DDS::TopicDescription pointer to a ***DDS::MultiTopic*** *(*p. 984*) pointer.*

6.118.1 Detailed Description

[**Not supported (optional)**] <<*interface*>> *(p. 175)* A specialization of DDS::TopicDescription that allows subscriptions that combine/filter/rearrange data coming from several topics.

DDS::MultiTopic (p. 984) 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. 433) entities associated with a **DDS::MultiTopic** (p. 984) are alerted of data modifications by the usual **DDS::Listener** (p. 952) or **DDS::WaitSet** (p. 1411) / **DDS::Condition** (p. 408) mechanisms whenever modifications occur to the data associated with any of the topics relevant to the **DDS::MultiTopic** (p. 984).

Note that the source for data may not be restricted to a single topic.

DDS::DataReader (p. 433) entities associated with a **DDS::MultiTopic** (p. 984) may access instances that are "constructed" at the **DDS::DataReader** (p. 433) side from the instances written by multiple **DDS::DataWriter** (p. 499) entities. The **DDS::MultiTopic** (p. 984) access instance will begin to exist as soon as all the constituting **DDS::Topic** (p. 1258) 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. 984) instance is `DDS::ViewStateKind::NEW_VIEW_STATE` (p. 1410) if at least one of the constituting instances has `view_state` =

DDS::ViewStateKind::NEW_VIEW_STATE (p. 1410). Otherwise, it will be **DDS::ViewStateKind::NOT_NEW_VIEW_STATE** (p. 1410).

^ The `instance_state` of the **DDS::MultiTopic** (p. 984) instance is **DDS::InstanceStateKind::ALIVE_INSTANCE_STATE** (p. 908) if the `instance_state` of all the constituting **DDS::Topic** (p. 1258) instances is **DDS::InstanceStateKind::ALIVE_INSTANCE_STATE** (p. 908). It is **DDS::InstanceStateKind::NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 909) if at least one of the constituting **DDS::Topic** (p. 1258) instances is **DDS::InstanceStateKind::NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 909). Otherwise, it is **DDS::InstanceStateKind::NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 909).

Queries and Filters Syntax (p. 184) describes the syntax of `subscription_expression` and `expression_parameters`.

6.118.2 Member Function Documentation

6.118.2.1 static MultiTopic ^ DDS::MultiTopic::narrow (ITopicDescription^ *topic_description*) [inline, static]

Narrow the given DDS::TopicDescription pointer to a **DDS::MultiTopic** (p. 984) pointer.

Returns:

DDS::MultiTopic (p. 984) if this DDS::TopicDescription is a **DDS::MultiTopic** (p. 984). Otherwise, return NULL.

6.118.2.2 System::String ^ DDS::MultiTopic::get_subscription_expression () [inline]

Get the expression for this **DDS::MultiTopic** (p. 984).

The expressions syntax is described in the DDS specification. It is specified when the **DDS::MultiTopic** (p. 984) is created.

Returns:

`subscription_expression` of the **DDS::MultiTopic** (p. 984).

6.118.2.3 void DDS::MultiTopic::get_expression_parameters (StringSeq^ *parameters*) [inline]

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. 987), or if **DDS::MultiTopic::set_expression_parameters** (p. 987) was never called, the `parameters` specified when the **DDS::MultiTopic** (p. 984) was created.

Parameters:

parameters *<<inout>>* *(p. 176)* Fill in this sequence with the expression parameters. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.118.2.4 void DDS::MultiTopic::set_expression_parameters (StringSeq^ *parameters*) [inline]

Set the `expression_parameters`.

Changes the `expression_parameters` associated with the **DDS::MultiTopic** (p. 984).

Parameters:

parameters *<<in>>* *(p. 175)* the filter expression parameters

Returns:

One of the **Standard Return Codes** (p. 235).

6.118.2.5 virtual System::String ^ DDS::MultiTopic::get_type_name () [inline, virtual]

Get the associated `type_name`.

The type name defines a locally unique type for the publication or the subscription.

The `type_name` corresponds to a unique string used to register a type via the **FooTypeSupport::register_type** (p. 885) method.

Thus, the `type_name` implies an association with a corresponding **DDS::TypeSupport** (p. 1385) and this DDS::TopicDescription.

Returns:

the type name. The returned type name is valid until the DDS::TopicDescription is deleted.

Postcondition:

The result is non-NULL.

See also:

DDS::TypeSupport (p. 1385), **FooTypeSupport** (p. 884)

Implements **DDS::ITopicDescription** (p. 913).

6.118.2.6 virtual System::String ^ DDS::MultiTopic::get_name () [inline, virtual]

Get the name used to create this DDS::TopicDescription .

Returns:

the name used to create this DDS::TopicDescription. The returned topic name is valid until the DDS::TopicDescription is deleted.

Postcondition:

The result is non-NULL.

Implements **DDS::ITopicDescription** (p. 913).

6.118.2.7 virtual DomainParticipant ^ DDS::MultiTopic::get_participant () [inline, virtual]

Get the **DDS::DomainParticipant** (p. 577) to which the DDS::TopicDescription belongs.

Returns:

The **DDS::DomainParticipant** (p. 577) to which the DDS::TopicDescription belongs.

Postcondition:

The result is non-NULL.

Implements **DDS::ITopicDescription** (p. 914).

6.119 DDS::OfferedDeadlineMissedStatus Struct Reference

DDS::StatusKind::OFFERED_DEADLINE_MISSED_STATUS

`#include <managed_publication.h>`

Public Attributes

- System::Int32 **total_count**

  *Total cumulative count of the number of times the **DDS::DataWriter** (p. 499) failed to write within its offered deadline.*

- System::Int32 **total_count_change**

  The incremental changes in total_count since the last time the listener was called or the status was read.

- **InstanceHandle_t last_instance_handle**

  *Handle to the last instance in the **DDS::DataWriter** (p. 499) for which an offered deadline was missed.*

6.119.1 Detailed Description

DDS::StatusKind::OFFERED_DEADLINE_MISSED_STATUS

Entity:

DDS::DataWriter (p. 499)

Listener:

DDS::DataWriterListener (p. 524)

The deadline that the **DDS::DataWriter** (p. 499) has committed through its **DDS::DeadlineQosPolicy** (p. 557) was not respected for a specific instance.

6.119.2 Member Data Documentation

6.119.2.1 System::Int32 DDS::OfferedDeadlineMissedStatus::total_count

Total cumulative count of the number of times the **DDS::DataWriter** (p. 499) failed to write within its offered deadline.

Missed deadlines accumulate; that is, each deadline period the `total_count` will be incremented by one.

6.119.2.2 System::Int32 DDS::OfferedDeadlineMissedStatus::total_count_change

The incremental changes in total_count since the last time the listener was called or the status was read.

6.119.2.3 InstanceHandle_t DDS::OfferedDeadlineMissedStatus::last_instance_handle

Handle to the last instance in the **DDS::DataWriter** (p. 499) for which an offered deadline was missed.

6.120 DDS::OfferedIncompatibleQosStatus Class Reference

DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS

`#include <managed_publication.h>`

Public Attributes

- System::Int32 **total_count**

  *Total cumulative number of times the concerned **DDS::DataWriter** (p. 499) discovered a **DDS::DataReader** (p. 433) for the same **DDS::Topic** (p. 1258), common partition with a requested QoS that is incompatible with that offered by the **DDS::DataWriter** (p. 499).*

- System::Int32 **total_count_change**

  The incremental changes in total_count since the last time the listener was called or the status was read.

- **QosPolicyId_t last_policy_id**

  The DDS::QosPolicyId_t of one of the policies that was found to be incompatible the last time an incompatibility was detected.

- **QosPolicyCountSeq^ policies**

  *A list containing for each policy the total number of times that the concerned **DDS::DataWriter** (p. 499) discovered a **DDS::DataReader** (p. 433) for the same **DDS::Topic** (p. 1258) and common partition with a requested QoS that is incompatible with that offered by the **DDS::DataWriter** (p. 499).*

6.120.1 Detailed Description

DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS

Entity:

DDS::DataWriter (p. 499)

Listener:

DDS::DataWriterListener (p. 524)

The qos policy value was incompatible with what was requested.

6.120.2 Member Data Documentation

6.120.2.1 System::Int32 DDS::OfferedIncompatibleQosStatus::total_count

Total cumulative number of times the concerned **DDS::DataWriter** (p. 499) discovered a **DDS::DataReader** (p. 433) for the same **DDS::Topic** (p. 1258), common partition with a requested QoS that is incompatible with that offered by the **DDS::DataWriter** (p. 499).

6.120.2.2 System::Int32 DDS::OfferedIncompatibleQosStatus::total_count_change

The incremental changes in total_count since the last time the listener was called or the status was read.

6.120.2.3 QosPolicyId_t DDS::OfferedIncompatibleQosStatus::last_policy_id

The DDS::QosPolicyId_t of one of the policies that was found to be incompatible the last time an incompatibility was detected.

6.120.2.4 QosPolicyCountSeq ^ DDS::OfferedIncompatibleQosStatus::policies

A list containing for each policy the total number of times that the concerned **DDS::DataWriter** (p. 499) discovered a **DDS::DataReader** (p. 433) for the same **DDS::Topic** (p. 1258) and common partition with a requested QoS that is incompatible with that offered by the **DDS::DataWriter** (p. 499).

6.121 DDS::OwnershipQosPolicy Struct Reference

Specifies whether it is allowed for multiple **DDS::DataWriter** (p. 499) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.

`#include <managed_infrastructure.h>`

Static Public Member Functions

^ static System::String^ **get_ownership_qos_policy_name** ()

*Stringified human-readable name for **DDS::OwnershipQosPolicy** (p. 993).*

Public Attributes

^ **OwnershipQosPolicyKind kind**

The kind of ownership.

6.121.1 Detailed Description

Specifies whether it is allowed for multiple **DDS::DataWriter** (p. 499) (s) to write the same instance of the data and if so, how these modifications should be arbitrated.

Entity:

DDS::Topic (p. 1258), **DDS::DataReader** (p. 433), **DDS::DataWriter** (p. 499)

Status:

DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS,
DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS

Properties:

RxO (p. 268) = YES
Changeable (p. 269) = **UNTIL ENABLE** (p. 269)

See also:

OWNERSHIP_STRENGTH (p. 285)

6.121.2 Usage

Along with the **OWNERSHIP_STRENGTH** (p. 285), this QoS policy specifies if **DDS::DataReader** (p. 433) entities can receive updates to the same instance (identified by its key) from multiple **DDS::DataWriter** (p. 499) entities at the same time.

There are two kinds of ownership, selected by the setting of the `kind:` SHARED and EXCLUSIVE.

6.121.2.1 SHARED ownership

DDS::OwnershipQosPolicyKind::SHARED_OWNERSHIP_QOS indicates that RTI Data Distribution Service 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. 1258) will be able to access modifications from all **DDS::DataWriter** (p. 499) objects, subject to the settings of other QoS that may filter particular samples (e.g. the **TIME_BASED_FILTER** (p. 288) or **HISTORY** (p. 294) policy). In any case, there is no "filtering" of modifications made based on the identity of the **DDS::DataWriter** (p. 499) that causes the modification.

6.121.2.2 EXCLUSIVE ownership

DDS::OwnershipQosPolicyKind::EXCLUSIVE_OWNERSHIP_QOS indicates that each instance of a data type can only be modified by one **DDS::DataWriter** (p. 499). In other words, at any point in time, a single **DDS::DataWriter** (p. 499) owns each instance and is the only one whose modifications will be visible to the **DDS::DataReader** (p. 433) objects. The owner is determined by selecting the **DDS::DataWriter** (p. 499) with the highest value of the **DDS::OwnershipStrengthQosPolicy::value** (p. 1001) that is currently alive, as defined by the **LIVELINESS** (p. 286) policy, and has not violated its **DEADLINE** (p. 281) contract with regards to the data instance.

Ownership can therefore change as a result of:

- a **DDS::DataWriter** (p. 499) 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. 499) that owns the instance, and
- a change in the liveliness of the **DDS::DataWriter** (p. 499) that owns the instance.

- a deadline with regards to the instance that is missed by the **DDS::DataWriter** (p. 499) that owns the instance.

The behavior of the system is as if the determination was made independently by each **DDS::DataReader** (p. 433). Each **DDS::DataReader** (p. 433) 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. 433) objects for that **DDS::Topic** (p. 1258) have a consistent picture of who owns each instance.

It is also not a requirement that the **DDS::DataWriter** (p. 499) objects are aware of whether they own a particular instance. There is no error or notification given to a **DDS::DataWriter** (p. 499) 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. 499) objects with the same strength modify the same instance. If this occurs RTI Data Distribution Service will pick one of the **DDS::DataWriter** (p. 499) 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. 433) objects will make the same choice of the particular **DDS::DataWriter** (p. 499) 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. 499) with higher same strength, or a new **DDS::DataWriter** (p. 499) 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. 499) as long as they affect instances whose values have not been set by the higher-strength **DDS::DataWriter** (p. 499).

6.121.3 Compatibility

The value of the DDS::OwnershipQosPolicyKind offered must exactly match the one requested or else they are considered incompatible.

6.121.4 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 **DDS::TypedDataWriter::unregister_instance** (p. 1372) and **DDS::TypedDataWriter::dispose** (p. 1379).

6.121.4.1 Ownership Resolution on Redundant Systems

It is expected that users may use DDS to set up redundant systems where multiple **DDS::DataWriter** (p. 499) entities are "capable" of writing the same instance. In this situation, the **DDS::DataWriter** (p. 499) 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::OwnershipQosPolicyKind::EXCLUSIVE_OWNERSHIP_QOS and arbitrate themselves by means of the **DDS::OwnershipStrengthQosPolicy** (p. 1000). Regardless of the scheme, the desired behavior from the **DDS::DataReader** (p. 433) point of view is that **DDS::DataReader** (p. 433) normally receives data from the primary unless the "primary" writer stops writing, in which case the **DDS::DataReader** (p. 433) starts to receive data from the secondary **DDS::DataWriter** (p. 499).

This approach requires some mechanism to detect that a **DDS::DataWriter** (p. 499) (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 **DDS::TypedDataWriter::write** (p. 1376).

Arbitrating from a **DDS::DataWriter** (p. 499) to one of a higher strength is simple and the decision can be taken autonomously by the **DDS::DataReader**

(p. 433). Switching ownership from a higher strength **DDS::DataWriter** (p. 499) to one of a lower strength **DDS::DataWriter** (p. 499) requires that the **DDS::DataReader** (p. 433) can make a determination that the stronger **DDS::DataWriter** (p. 499) is "no longer writing the instance".

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. 499) simply states its offered **DDS::DeadlineQosPolicy** (p. 557) (maximum interval between updates) and the **DDS::DataReader** (p. 433) automatically monitors that the **DDS::DataWriter** (p. 499) indeed updates the instance at least once per **DDS::DeadlineQosPolicy::period** (p. 559). If the deadline is missed, the **DDS::DataReader** (p. 433) considers the **DDS::DataWriter** (p. 499) "not alive" and automatically gives ownership to the next highest-strength **DDS::DataWriter** (p. 499) that *is* alive.

Case where data is not periodically updated The case where the **DDS::DataWriter** (p. 499) 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. 499) is "alive" and for the **DDS::DataWriter** (p. 499) to be alive it must fulfill its **DDS::LivelinessQosPolicy** (p. 960) 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::LivelinessQosPolicyKind::MANUAL_BY_TOPIC_LIVELINESS_QOS. The **DDS::DataWriter** (p. 499) 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. 499) process periodically writes data or calls **DDS::DomainParticipant::assert_liveliness** (p. 637). However, this scenario requires that the **DDS::DataReader** (p. 433) knows what instances are being "written" by the **DDS::DataWriter** (p. 499). That is the only way that the **DDS::DataReader** (p. 433) deduces the ownership of specific instances from the fact that the **DDS::DataWriter** (p. 499) is still "alive". Hence the need for the **DDS::DataWriter** (p. 499) to "register" and "unregister" instances. Note that while "registration" can be done lazily the first time the **DDS::DataWriter** (p. 499) 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. 433).

6.121.4.2 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. 557) 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. 960) in combination with register_instance/unregister_instance to detect whether the "connectivity" has been lost, and the notification is provided by means of **DDS::InstanceStateKind::NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 909).

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).

6.121.4.3 Semantic Difference between unregister_instance and dispose

DDS::TypedDataWriter::dispose (p. 1379) is semantically different from **DDS::TypedDataWriter::unregister_instance** (p. 1372). **DDS::TypedDataWriter::dispose** (p. 1379) 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 **DDS::TypedDataWriter::unregister_instance** (p. 1372) indicates that the writer is no longer taking responsibility for updating the value of the instance.

Deleting a **DDS::DataWriter** (p. 499) 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. 1258) with DDS::OwnershipQosPolicyKind::EXCLUSIVE_-

OWNERSHIP_QOS, 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. 499) 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. 1258) with DDS::OwnershipQosPolicyKind::EXCLUSIVE_OWNERSHIP_QOS, 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.

6.121.5 Member Data Documentation

6.121.5.1 OwnershipQosPolicyKind DDS::OwnershipQosPolicy::kind

The kind of ownership.

[**default**] DDS::OwnershipQosPolicyKind::SHARED_OWNERSHIP_QOS

6.122 DDS::OwnershipStrengthQosPolicy Struct Reference

Specifies the value of the strength used to arbitrate among multiple **DDS::DataWriter** (p. 499) objects that attempt to modify the same instance of a data type (identified by **DDS::Topic** (p. 1258) + key).

`#include <managed_infrastructure.h>`

Static Public Member Functions

^ static System::String^ **get_ownershipstrength_qos_policy_name** ()

*Stringified human-readable name for **DDS::OwnershipStrengthQosPolicy** (p. 1000).*

Public Attributes

^ System::Int32 **value**

The strength value used to arbitrate among multiple writers.

6.122.1 Detailed Description

Specifies the value of the strength used to arbitrate among multiple **DDS::DataWriter** (p. 499) objects that attempt to modify the same instance of a data type (identified by **DDS::Topic** (p. 1258) + key).

This policy only applies if the **OWNERSHIP** (p. 283) policy is of kind DDS::OwnershipQosPolicyKind::EXCLUSIVE_OWNERSHIP_QOS.

Entity:

DDS::DataWriter (p. 499)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **YES** (p. 269)

The value of the **OWNERSHIP_STRENGTH** (p. 285) is used to determine the ownership of a data instance (identified by the key). The arbitration is performed by the **DDS::DataReader** (p. 433).

See also:

EXCLUSIVE ownership (p. 994)

6.122.2 Member Data Documentation

6.122.2.1 System::Int32 DDS::OwnershipStrengthQosPolicy::value

The strength value used to arbitrate among multiple writers.

[default] 0

[range] [0, 1 million]

6.123 DDS::ParticipantBuiltinTopicData Class Reference

Entry created when a **DomainParticipant** (p. 577) object is discovered.

`#include <managed_builtin.h>`

Inheritance diagram for DDS::ParticipantBuiltinTopicData::

Public Attributes

- **BuiltinTopicKey_t key**

  DCPS key to distinguish entries.

- **ProtocolVersion_t rtps_protocol_version**

  <<**eXtension**>> (p. 174) *Version number of the RTPS wire protocol used.*

- **VendorId_t rtps_vendor_id**

  <<**eXtension**>> (p. 174) *ID of vendor implementing the RTPS wire protocol.*

- System::UInt32 **dds_builtin_endpoints**

  <<**eXtension**>> (p. 174) *Bitmap of builtin endpoints supported by the participant.*

- **ProductVersion_t product_version**

  <<**eXtension**>> (p. 174) *This is a vendor specific parameter. It gives the current version for rti-dds.*

Properties

- **UserDataQosPolicy^ user_data** `[get]`

  *Policy of the corresponding **DomainParticipant*** (p. 577).

- **PropertyQosPolicy^ property_qos** `[get]`

  <<**eXtension**>> (p. 174) *Name value pair properties to be stored with domain participant*

- **LocatorSeq^ default_unicast_locators** `[get]`

  <<**eXtension**>> (p. 174) *Unicast locators used when individual entities do not specify unicast locators.*

^ **EntityNameQosPolicy^ participant_name** [get]

<<**eXtension**>> (p. 174) *The participant name.*

6.123.1 Detailed Description

Entry created when a **DomainParticipant** (p. 577) object is discovered.

Data associated with the built-in topic **DDS::ParticipantBuiltinTopicDataTypeSupport::PARTICIPANT_TOPIC_NAME** (p. 226). It contains QoS policies and additional information that apply to the remote **DDS::DomainParticipant** (p. 577).

See also:

DDS::ParticipantBuiltinTopicDataTypeSupport::PARTICIPANT_TOPIC_NAME (p. 226)
DDS::ParticipantBuiltinTopicDataDataReader (p. 1005)

6.123.2 Member Data Documentation

6.123.2.1 BuiltinTopicKey_t DDS::ParticipantBuiltinTopicData::key

DCPS key to distinguish entries.

6.123.2.2 ProtocolVersion_t DDS::ParticipantBuiltinTopicData::rtps_protocol_version

<<***eXtension***>> *(p. 174)* Version number of the RTPS wire protocol used.

6.123.2.3 VendorId_t DDS::ParticipantBuiltinTopicData::rtps_vendor_id

<<***eXtension***>> *(p. 174)* ID of vendor implementing the RTPS wire protocol.

6.123.2.4 System::UInt32 DDS::ParticipantBuiltinTopicData::dds_builtin_endpoints

<<***eXtension***>> *(p. 174)* Bitmap of builtin endpoints supported by the participant.

Each bit indicates a builtin endpoint that may be available on the participant for use in discovery.

6.123.2.5 ProductVersion_t DDS::ParticipantBuiltinTopicData::product_version

<<*eXtension*>> *(p. 174)* This is a vendor specific parameter. It gives the current version for rti-dds.

6.123.3 Property Documentation

6.123.3.1 UserDataQosPolicy^ DDS::ParticipantBuiltinTopicData::user_data [get]

Policy of the corresponding **DomainParticipant** (p. 577).

6.123.3.2 PropertyQosPolicy^ DDS::ParticipantBuiltinTopicData::property_qos [get]

<<*eXtension*>> *(p. 174)* Name value pair properties to be stored with domain participant

6.123.3.3 LocatorSeq^ DDS::ParticipantBuiltinTopicData::default_unicast_locators [get]

<<*eXtension*>> *(p. 174)* Unicast locators used when individual entities do not specify unicast locators.

6.123.3.4 EntityNameQosPolicy^ DDS::ParticipantBuiltinTopicData::participant_name [get]

<<*eXtension*>> *(p. 174)* The participant name.

This is the name of the discovered participant.

6.124 DDS::ParticipantBuiltinTopicDataDataReader Class Reference

Instantiates `DataReader` (p. 433) < **DDS::ParticipantBuiltinTopicData** (p. 1002) > .

`#include <managed_builtin.h>`

Inheritance diagram for DDS::ParticipantBuiltinTopicDataDataReader::

6.124.1 Detailed Description

Instantiates `DataReader` (p. 433) < **DDS::ParticipantBuiltinTopicData** (p. 1002) > .

DDS::DataReader (p. 433) of topic **DDS::ParticipantBuiltinTopicDataTypeSupport::PARTICIPANT_TOPIC_NAME** (p. 226) used for accessing **DDS::ParticipantBuiltinTopicData** (p. 1002) of the remote **DDS::DomainParticipant** (p. 577).

Instantiates:

<<*generic*>> *(p. 175)* **DDS::TypedDataReader** (p. 1338)

See also:

DDS::ParticipantBuiltinTopicData (p. 1002)
DDS::ParticipantBuiltinTopicDataTypeSupport::PARTICIPANT_TOPIC_NAME (p. 226)

6.125 DDS::ParticipantBuiltinTopicDataSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < **DDS::ParticipantBuiltinTopicData** (p. 1002) > .

`#include <managed_builtin.h>`

6.125.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < **DDS::ParticipantBuiltinTopicData** (p. 1002) > .

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

DDS::ParticipantBuiltinTopicData (p. 1002)

6.126 DDS::ParticipantBuiltinTopicDataTypeSupport Class Reference

Instantiates `TypeSupport` (p. 1385) < **DDS::ParticipantBuiltinTopicData** (p. 1002) > .

`#include <managed_builtin.h>`

Inherits DDS::AbstractBuiltinTopicDataTypeSupport< T >.

Properties

^ static System::String^ **PARTICIPANT_TOPIC_NAME** `[get]`

Participant topic name.

6.126.1 Detailed Description

Instantiates `TypeSupport` (p. 1385) < **DDS::ParticipantBuiltinTopicData** (p. 1002) > .

Instantiates:

<<***generic***>> *(p. 175)* **FooTypeSupport** (p. 884)

See also:

DDS::ParticipantBuiltinTopicData (p. 1002)

6.127 DDS::PartitionQosPolicy Class Reference

Set of strings that introduces a logical partition among the topics visible by a **DDS::Publisher** (p. 1044) and a **DDS::Subscriber** (p. 1201).

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_partition_qos_policy_name** ()

  *Stringified human-readable name for **DDS::PartitionQosPolicy** (p. 1008).*

Public Attributes

- **StringSeq^ name**

  A list of partition names.

6.127.1 Detailed Description

Set of strings that introduces a logical partition among the topics visible by a **DDS::Publisher** (p. 1044) and a **DDS::Subscriber** (p. 1201).

This QoS policy is used to set string identifiers that are used for matching DataReaders and DataWriters for the same **Topic** (p. 1258).

A **DDS::DataWriter** (p. 499) within a **DDS::Publisher** (p. 1044) only communicates with a **DDS::DataReader** (p. 433) in a **DDS::Subscriber** (p. 1201) if (in addition to matching the **DDS::Topic** (p. 1258) and having compatible QoS) the **DDS::Publisher** (p. 1044) and **DDS::Subscriber** (p. 1201) have a common partition name string.

Entity:

DDS::Publisher (p. 1044), **DDS::Subscriber** (p. 1201)

Properties:

RxO (p. 268) = NO
Changeable (p. 269) = **YES** (p. 269)

6.127.2 Usage

This policy allows the introduction of a logical partition concept inside the 'physical' partition induced by a *domain*.

Usually DataReaders and DataWriters are matched only by their topic (so that data are only sent by DataWriters to DataReaders for the same topic). The Partition QoS policy allows you to add one or more strings, "partitions", to a **Publisher** (p. 1044) and/or **Subscriber** (p. 1201). If partitions are added, then a **DataWriter** (p. 499) and **DataReader** (p. 433) for the same topic are only considered matched if their Publishers and Subscribers have partitions in common (intersecting partitions).

Since the set of partitions for a publisher or subscriber can be dynamically changed, the Partition QoS policy is useful to control which DataWriters can send data to which DataReaders and vice versa - even if all of the DataWriters and DataReaders are for the same topic. This facility is useful for creating temporary separation groups among entities that would otherwise be connected to and exchange data each other.

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** (p. 433) and **DataWriter** (p. 499) 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 a **Publisher** (p. 1044) intersect with the partitions or wildcard patterns of a **Subscriber** (p. 1201), their DataWriters and DataReaders of the same topic are able to match; otherwise they are not.

These partition name patterns are regular expressions as defined by the POSIX fnmatch API (1003.2-1992 section B.6). Either **DDS::Publisher** (p. 1044) or **DDS::Subscriber** (p. 1201) 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 both at publisher as well as subscriber side, RTI Data Distribution Service will not try to match two regular expressions (between publishers and subscribers).

Each publisher and subscriber must belong to at least one logical partition. A regular expression is not considered to be a logical partition. If a publisher or subscriber has not specify 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. 845) 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 Data Distribution Service itself to see entities in a domain it does not belong to.
- Second, a **DDS::Entity** (p. 845) can only belong to one domain whereas a **DDS::Entity** (p. 845) can be in multiple partitions.
- Finally, as far as RTI Data Distribution Service is concerned, each unique data instance is identified by the tuple **(DomainID, DDS::Topic** (p. **1258**)**, key)**. Therefore two **DDS::Entity** (p. 845) 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.

6.127.3 Member Data Documentation

6.127.3.1 StringSeq ^ DDS::PartitionQosPolicy::name

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_InconsistentPolicy** (p. 1119) when setting a **DDS::Publisher** (p. 1044)'s or **DDS::Subscriber** (p. 1201)'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. 1008) is specified on a domain basis in **DDS::DomainParticipantResourceLimitsQosPolicy::max_partitions** (p. 702). 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. 1008) is specified on a domain basis in **DDS::DomainParticipantResourceLimitsQosPolicy::max_partition_cumulative_characters** (p. 703).

[**default**] Empty sequence (zero-length sequence). Since no logical partition is specified, RTI Data Distribution Service will assume the entity to be in default partition (empty string partition "").

[**range**] List of partition name with above restrictions

6.128 DDS::PresentationQosPolicy Struct Reference

Specifies how the samples representing changes to data instances are presented to a subscribing application.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_presentation_qos_policy_name** ()

  Stringified human-readable name for ***DDS::PresentationQosPolicy*** *(*p. 1012*).*

Public Attributes

- **PresentationQosPolicyAccessScopeKind access_scope**

  Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.

Properties

- System::Boolean **coherent_access** `[get, set]`

  Specifies support for coherent *access. Controls whether coherent access is supported within the scope* `access_scope`*.*

- System::Boolean **ordered_access** `[get, set]`

  Specifies support for ordered *access to the samples received at the subscription end. Controls whether ordered access is supported within the scope* `access_scope`*.*

6.128.1 Detailed Description

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 Data Distribution Service. 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. 1044), **DDS::Subscriber** (p. 1201)

Status:

DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS,
DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS

Properties:

RxO (p. 268) = YES
Changeable (p. 269) = **UNTIL ENABLE** (p. 269)

6.128.2 Usage

A **DDS::DataReader** (p. 433) will usually receive data in the order that it was sent by a **DDS::DataWriter** (p. 499), and the data is presented to the **DDS::DataReader** (p. 433) as soon as the application receives the next expected value. However, sometimes, you may want a set of data for the same topic to be presented to the **DDS::DataReader** (p. 433) only after *all* of the elements of the set have been received. Or you may want the data to be presented in a different order than that in which it was received. 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 allows you to specify different *scopes* of presentation: within a topic, across instances of a single topic, and even across multiple topics used by different writers of a publisher (although this last option is not currently supported by RTI). 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 Data Distribution Service will preserve the groupings of changes made by a publishing application by means of the operations **DDS::Publisher::begin_coherent_changes** (p. 1060) and **DDS::Publisher::end_coherent_changes** (p. 1060).
- `ordered_access` controls whether RTI Data Distribution Service 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` controls the maximum extent of coherent changes. The behavior is as follows:

- If `access_scope` is set to DDS::PresentationQosPolicyAccessScopeKind::INSTANCE_PRESENTATION_QOS (the default), the use of **DDS::Publisher::begin_coherent_changes** (p. 1060) and **DDS::Publisher::end_coherent_changes** (p. 1060) has no effect on how the subscriber can access the data, because with the scope limited to each instance, changes to separate instances are considered independent and thus cannot be grouped into a coherent set.

- If `access_scope` is set to DDS::PresentationQosPolicyAccessScopeKind::TOPIC_PRESENTATION_QOS, then coherent changes (indicated by their enclosure within calls to **DDS::Publisher::begin_coherent_changes** (p. 1060) and **DDS::Publisher::end_coherent_changes** (p. 1060)) will be made available as such to each remote **DDS::DataReader** (p. 433) independently. That is, changes made to instances within each individual **DDS::DataWriter** (p. 499) will be available as coherent with respect to other changes to instances in that same **DDS::DataWriter** (p. 499), but will not be grouped with changes made to instances belonging to a different **DDS::DataWriter** (p. 499).

- If access_scope is set to DDS::PresentationQosPolicyAccessScopeKind::GROUP_PRESENTATION_QOS, then coherent changes made to instances through a **DDS::DataWriter** (p. 499) attached to a common **DDS::Publisher** (p. 1044) are made available as a unit to remote subscribers. *(RTI does not currently support this access scope.)*

If `ordered_access` is set, then the `access_scope` controls the maximum extent for which order will be preserved by RTI Data Distribution Service.

- If `access_scope` is set to DDS::PresentationQosPolicyAccessScopeKind::INSTANCE_PRESENTATION_QOS (the lowest level), then changes to each instance are considered unordered relative to changes to any other instance. That means that changes (creations, deletions, modifications) made to two instances are not necessarily seen in the order they occur. This is the case even if it is the same application thread making the changes using the same **DDS::DataWriter** (p. 499).

- If `access_scope` is set to DDS::PresentationQosPolicyAccessScopeKind::TOPIC_PRESENTATION_QOS, changes (creations, deletions, modifications) made by a single **DDS::DataWriter** (p. 499) are made available to subscribers in the same order they occur. Changes made to instances though different **DDS::DataWriter** (p. 499) entities are not necessarily

seen in the order they occur. This is the case, even if the changes are made by a single application thread using **DDS::DataWriter** (p. 499) objects attached to the same **DDS::Publisher** (p. 1044).

- Finally, if access_scope is set to DDS::PresentationQosPolicyAccessScopeKind::GROUP_PRESENTATION_QOS, changes made to instances via **DDS::DataWriter** (p. 499) entities attached to the same **DDS::Publisher** (p. 1044) object are made available to subscribers on the same order they occur. *(RTI does not currently support this access scope.)*

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. 1201) **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. 433) objects.

6.128.3 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'. For the purposes of this inequality, the values of `access_scope` are considered ordered such that DDS::PresentationQosPolicyAccessScopeKind::INSTANCE_PRESENTATION_QOS < DDS::PresentationQosPolicyAccessScopeKind::TOPIC_PRESENTATION_QOS < DDS::PresentationQosPolicyAccessScopeKind::GROUP_PRESENTATION_QOS
- requested `coherent_access` is false, or else both offered and requested `coherent_access` are true
- requested `ordered_access` is false, or else both offered and requested `ordered_access` are true.

6.128.4 Member Data Documentation

6.128.4.1 PresentationQosPolicyAccessScopeKind DDS::PresentationQosPolicy::access_scope

Determines the largest scope spanning the entities for which the order and coherency of changes can be preserved.

[default] DDS::PresentationQosPolicyAccessScopeKind::INSTANCE_-PRESENTATION_QOS

6.128.5 Property Documentation

6.128.5.1 System:: Boolean DDS::PresentationQosPolicy::coherent_-access [get, set]

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.

[default] false

6.128.5.2 System:: Boolean DDS::PresentationQosPolicy::ordered_-access [get, set]

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] false

6.129 DDS::ProductVersion_t Struct Reference

<<*eXtension*>> *(p. 174)* Type used to represent the current version of RTI Data Distribution Service.

`#include <managed_infrastructure.h>`

Public Attributes

- System::SByte **major**

  Major product version.

- System::SByte **minor**

  Minor product version.

- System::SByte **release**

  Release letter for product version.

- System::SByte **revision**

  Revision number of product.

Properties

- static **ProductVersion_t** **PRODUCTVERSION_UNKNOWN** `[get]`

  The value used when the product version is unknown.

6.129.1 Detailed Description

<<*eXtension*>> *(p. 174)* Type used to represent the current version of RTI Data Distribution Service.

6.129.2 Member Data Documentation

6.129.2.1 System::SByte DDS::ProductVersion_t::major

Major product version.

6.129.2.2 System::SByte DDS::ProductVersion_t::minor

Minor product version.

6.129.2.3 System::SByte DDS::ProductVersion_t::release

Release letter for product version.

6.129.2.4 System::SByte DDS::ProductVersion_t::revision

Revision number of product.

6.130 DDS::ProfileQosPolicy Class Reference

Configures the way that XML documents containing QoS profiles are loaded by RTI Data Distribution Service.

`#include <managed_infrastructure.h>`

Static Public Member Functions

^ static System::String^ **get_profile_qos_policy_name** ()

*Stringified human-readable name for **DDS::ProfileQosPolicy** (p. 1019).*

Public Attributes

^ **StringSeq^ string_profile**

***Sequence** (p. 1163) of strings containing a XML document to load.*

^ **StringSeq^ url_profile**

***Sequence** (p. 1163) of **URL groups** (p. 137) containing a set of XML documents to load.*

Properties

^ System::Boolean **ignore_user_profile** `[get, set]`

Ignores the file USER_QOS_PROFILES.xml in the current working directory.

^ System::Boolean **ignore_environment_profile** `[get, set]`

*Ignores the value of the **NDDS_QOS_PROFILES environment variable** (p. 137).*

^ System::Boolean **ignore_resource_profile** `[get, set]`

Ignores the file NDDS_QOS_PROFILES.xml under $NDDSHOME/resource/qos_profiles_4.4d/xml.

6.130.1 Detailed Description

Configures the way that XML documents containing QoS profiles are loaded by RTI Data Distribution Service.

All QoS values for Entities can be configured in QoS profiles defined in XML documents. XML documents can be passed to RTI Data Distribution Service 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 Data Distribution Service, 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. 649)

Properties:

RxO (p. 268) = NO
Changeable (p. 269) = **Changeable** (p. 269)

6.130.2 Member Data Documentation

6.130.2.1 StringSeq ^ DDS::ProfileQosPolicy::string_profile

Sequence (p. 1163) 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).

6.130.2.2 StringSeq ^ DDS::ProfileQosPolicy::url_profile

Sequence (p. 1163) of **URL groups** (p. 137) containing a set of XML documents to load.

Only one of the elements of each group will be loaded by RTI Data Distribution Service, starting from the left.

[**default**] Empty sequence (zero-length).

6.130.3 Property Documentation

6.130.3.1 System:: Boolean DDS::ProfileQosPolicy::ignore_user_profile [get, set]

Ignores the file USER_QOS_PROFILES.xml in the current working directory.

When this field is set to true, the QoS profiles contained in the file USER_QOS_PROFILES.xml in the current working directory will be ignored.

[**default**] false

6.130.3.2 System:: Boolean DDS::ProfileQosPolicy::ignore_environment_profile [get, set]

Ignores the value of the **NDDS_QOS_PROFILES environment variable** (p. 137).

When this field is set to true, the value of the environment variable NDDS_QOS_PROFILES will be ignored.

[**default**] false

6.130.3.3 System:: Boolean DDS::ProfileQosPolicy::ignore_resource_profile [get, set]

Ignores the file NDDS_QOS_PROFILES.xml under $NDDSHOME/resource/qos_profiles_4.4d/xml.

When this field is set to true, the QoS profiles contained in the file NDDS_QOS_PROFILES.xml under $NDDSHOME/resource/qos_profiles_4.5c/xml will be ignored.

[**default**] false

6.131 DDS::Property_t Class Reference

Properties are name/value pairs objects.

`#include <managed_infrastructure.h>`

Public Attributes

- ^ System::String^ **name**
  Property name.
- ^ System::String^ **value**
  Property value.

Properties

- ^ System::Boolean **propagate** `[get, set]`
  Indicates if the property must be propagated on discovery.

6.131.1 Detailed Description

Properties are name/value pairs objects.

6.131.2 Member Data Documentation

6.131.2.1 System::String ^ DDS::Property_t::name

Property name.

6.131.2.2 System::String ^ DDS::Property_t::value

Property value.

6.131.3 Property Documentation

6.131.3.1 System:: Boolean DDS::Property_t::propagate `[get, set]`

Indicates if the property must be propagated on discovery.

6.132 DDS::PropertyQosPolicy Class Reference

Stores name/value(string) pairs that can be used to configure certain parameters of RTI Data Distribution Service 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.

`#include <managed_infrastructure.h>`

Static Public Member Functions

^ static System::String^ **get_property_qos_policy_name** ()

*Stringified human-readable name for **DDS::PropertyQosPolicy** (p. 1023).*

Public Attributes

^ **PropertySeq^ value**

***Sequence** (p. 1163) of properties.*

6.132.1 Detailed Description

Stores name/value(string) pairs that can be used to configure certain parameters of RTI Data Distribution Service 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. 577) **DDS::DataReader** (p. 433) **DDS::DataWriter** (p. 499)

Properties:

RxO (p. 268) = N/A;
Changeable (p. 269) = **YES** (p. 269)

See also:

DDS::DomainParticipant::get_builtin_subscriber (p. 632)

6.132.2 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. 433), **DDS::DataWriter** (p. 499), or **DDS::DomainParticipant** (p. 577). This is similar to the **DDS::UserDataQosPolicy** (p. 1403), 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).

This QoS policy may be used to configure:

- Durable Writer History, see **Configuring Durable Writer History** (p. 131)
- Durable Reader State, see **Configuring Durable Reader State** (p. 131)
- Builtin Transport Plugins, see **UDPv4 Transport Property Names in Property QoS Policy of Domain Participant** (p. 1389), **UDPv6 Transport Property Names in Property QoS Policy of Domain Participant** (p. 1392), and **Shared Memory Transport Property Names in Property QoS Policy of Domain Participant** (p. 1179)
- Extension Transport Plugins, see **Loading Transport Plugins through Property QoS Policy of Domain Participant** (p. 119)
- **Clock Selection** (p. 31)

In addition, you may add your own name/value pairs to the Property QoS policy of an **Entity** (p. 845). Via this QoS policy, you can direct RTI Data Distribution Service to propagate these name/value pairs with the discovery information for the **Entity** (p. 845). Applications that discover the **Entity** (p. 845) can then access the user-specific name/value pairs in the discovery information of the remote **Entity** (p. 845). This allows you to add meta-information about an **Entity** (p. 845) for application-specific use, for example, authentication/authorization certificates (which can also be done using the **DDS::UserDataQosPolicy** (p. 1403) or **DDS::GroupDataQosPolicy** (p. 890)).

6.132.2.1 Reasons for Using the PropertyQosPolicy

- Supports dynamic loading of extension transports (such as RTI Secure WAN Transport)
- 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 Data Distribution Service 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** (p. 845) 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 **DDS::PropertyQosPolicyHelper** (p. 1026) class on the **PROPERTY** (p. 357) page.

6.132.3 Member Data Documentation

6.132.3.1 PropertySeq ^ DDS::PropertyQosPolicy::value

Sequence (p. 1163) of properties.

[**default**] An empty list.

6.133 DDS::PropertyQosPolicyHelper Class Reference

Policy Helpers which facilitate management of the properties in the input policy.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- ^ static Int32 **get_number_of_properties** (**PropertyQosPolicy**^ policy)

  Gets the number of properties in the input policy.

- ^ static void **assert_property** (**PropertyQosPolicy**^ policy, String^ name, String^ value, System::Boolean propagate)

  Asserts the property identified by name in the input policy.

- ^ static void **add_property** (**PropertyQosPolicy**^ policy, String^ name, String^ value, System::Boolean propagate)

  Adds a new property to the input policy.

- ^ static **Property_t**^ **lookup_property** (**PropertyQosPolicy**^ policy, System::String^ name)

  Searches for a property in the input policy given its name.

- ^ static void **remove_property** (**PropertyQosPolicy**^ policy, String^ name)

  Removes a property from the input policy.

- ^ static void **get_properties** (**PropertyQosPolicy**^ policy, **PropertySeq**^ properties, String^ name_prefix)

  Retrieves a list of properties whose names match the input prefix.

6.133.1 Detailed Description

Policy Helpers which facilitate management of the properties in the input policy.

6.134 DDS::PropertySeq Class Reference

Declares IDL `sequence` < **DDS::Property_t** (p. 1022) >.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::PropertySeq::

6.134.1 Detailed Description

Declares IDL `sequence` < **DDS::Property_t** (p. 1022) >.

See also:

DDS::Property_t (p. 1022)

6.135 DDS::ProtocolVersion_t Struct Reference

<<*eXtension*>> *(p. 174)* Type used to represent the version of the RTPS protocol.

`#include <managed_infrastructure.h>`

Public Attributes

- System::Byte **major**

  Major protocol version number.

- System::Byte **minor**

  Minor protocol version number.

Properties

- static **ProtocolVersion_t PROTOCOLVERSION** `[get]`

  The most recent protocol version. Currently 1.2.

- static **ProtocolVersion_t PROTOCOLVERSION_1_0** `[get]`

  The protocol version 1.0.

- static **ProtocolVersion_t PROTOCOLVERSION_1_1** `[get]`

  The protocol version 1.1.

- static **ProtocolVersion_t PROTOCOLVERSION_1_2** `[get]`

  The protocol version 1.2.

- static **ProtocolVersion_t PROTOCOLVERSION_2_0** `[get]`

  The protocol version 2.0.

- static **ProtocolVersion_t PROTOCOLVERSION_2_1** `[get]`

  The protocol version 2.1.

6.135.1 Detailed Description

<<*eXtension*>> *(p. 174)* Type used to represent the version of the RTPS protocol.

6.135.2 Member Data Documentation

6.135.2.1 System::Byte DDS::ProtocolVersion_t::major

Major protocol version number.

6.135.2.2 System::Byte DDS::ProtocolVersion_t::minor

Minor protocol version number.

6.136 DDS::PublicationBuiltinTopicData Class Reference

Entry created when a **DDS::DataWriter** (p. 499) is discovered in association with its **Publisher** (p. 1044).

`#include <managed_builtin.h>`

Inheritance diagram for DDS::PublicationBuiltinTopicData::

Public Attributes

- **BuiltinTopicKey_t key**

  DCPS key to distinguish entries.

- **BuiltinTopicKey_t participant_key**

  *DCPS key of the participant to which the **DataWriter** (p. 499) belongs.*

- System::String^ **topic_name**

  *Name of the related **DDS::Topic** (p. 1258).*

- System::String^ **type_name**

  *Name of the type attached to the **DDS::Topic** (p. 1258).*

- **DurabilityQosPolicy durability**

  *durability policy of the corresponding **DataWriter** (p. 499)*

- **DurabilityServiceQosPolicy durability_service**

  *durability_service policy of the corresponding **DataWriter** (p. 499)*

- **DeadlineQosPolicy deadline**

  *Policy of the corresponding **DataWriter** (p. 499).*

- **LatencyBudgetQosPolicy latency_budget**

  *Policy of the corresponding **DataWriter** (p. 499).*

- **LivelinessQosPolicy liveliness**

  *Policy of the corresponding **DataWriter** (p. 499).*

- **ReliabilityQosPolicy reliability**

  *Policy of the corresponding **DataWriter** (p. 499).*

^ **LifespanQosPolicy lifespan**

Policy of the corresponding ***DataWriter*** *(p. 499).*

^ **OwnershipQosPolicy ownership**

Policy of the corresponding ***DataWriter*** *(p. 499).*

^ **OwnershipStrengthQosPolicy ownership_strength**

Policy of the corresponding ***DataWriter*** *(p. 499).*

^ **DestinationOrderQosPolicy destination_order**

Policy of the corresponding ***DataWriter*** *(p. 499).*

^ **PresentationQosPolicy presentation**

Policy of the ***Publisher*** *(p. 1044) to which the* ***DataWriter*** *(p. 499) belongs.*

^ **BuiltinTopicKey_t publisher_key**

<<**eXtension**>> (p. 174) *DCPS key of the publisher to which the* ***DataWriter*** *(p. 499) belongs*

^ **GUID_t virtual_guid**

<<**eXtension**>> (p. 174) *Virtual GUID associated to the* ***DataWriter*** *(p. 499).*

^ **ProtocolVersion_t rtps_protocol_version**

<<**eXtension**>> (p. 174) *Version number of the RTPS wire protocol used.*

^ **VendorId_t rtps_vendor_id**

<<**eXtension**>> (p. 174) *ID of vendor implementing the RTPS wire protocol.*

^ **ProductVersion_t product_version**

<<**eXtension**>> (p. 174) *This is a vendor specific parameter. It gives the current version for rti-dds.*

^ **LocatorFilterQosPolicy^ locator_filter**

<<**eXtension**>> (p. 174) *Policy of the corresponding* ***DataWriter*** *(p. 499)*

Properties

^ **UserDataQosPolicy^ user_data** [get]

Policy of the corresponding ***DataWriter*** *(p. 499).*

^ **PartitionQosPolicy^ partition** [get]

Policy of the ***Publisher*** *(p. 1044) to which the* ***DataWriter*** *(p. 499) belongs.*

^ **TopicDataQosPolicy^ topic_data** [get]

Policy of the related ***Topic*** *(p. 1258).*

^ **GroupDataQosPolicy^ group_data** [get]

Policy of the ***Publisher*** *(p. 1044) to which the* ***DataWriter*** *(p. 499) belongs.*

^ **DDS::TypeCode^ type_code** [get]

<<**eXtension**>> (p. 174) *Type code information of the corresponding* ***Topic*** *(p. 1258)*

^ **PropertyQosPolicy^ property_qos** [get]

<<**eXtension**>> (p. 174) *Properties of the corresponding* ***DataWriter*** *(p. 499).*

^ **LocatorSeq^ unicast_locators** [get]

<<**eXtension**>> (p. 174) *Custom unicast locators that the endpoint can specify. The default locators will be used if this is not specified.*

^ System::Boolean **disable_positive_acks** [get, set]

<<**eXtension**>> (p. 174) *This is a vendor specific parameter. Determines whether matching DataReaders send positive acknowledgements for reliability.*

6.136.1 Detailed Description

Entry created when a **DDS::DataWriter** (p. 499) is discovered in association with its **Publisher** (p. 1044).

Data associated with the built-in topic **DDS::PublicationBuiltinTopicDataTypeSupport::PUBLIC**
TOPIC_NAME (p. 230). It contains QoS policies and additional information that apply to the remote **DDS::DataWriter** (p. 499) the related **DDS::Publisher** (p. 1044).

See also:

DDS::PublicationBuiltinTopicDataTypeSupport::PUBLICATION_TOPIC_NAME (p. 230)
DDS::PublicationBuiltinTopicDataDataReader (p. 1038)

6.136.2 Member Data Documentation

6.136.2.1 BuiltinTopicKey_t DDS::PublicationBuiltinTopicData::key

DCPS key to distinguish entries.

6.136.2.2 BuiltinTopicKey_t DDS::PublicationBuiltinTopicData::participant_key

DCPS key of the participant to which the **DataWriter** (p. 499) belongs.

6.136.2.3 System::String ^ DDS::PublicationBuiltinTopicData::topic_name

Name of the related **DDS::Topic** (p. 1258).

The length of this string is limited to 255 characters.

6.136.2.4 System::String ^ DDS::PublicationBuiltinTopicData::type_name

Name of the type attached to the **DDS::Topic** (p. 1258).

The length of this string is limited to 255 characters.

6.136.2.5 DurabilityQosPolicy DDS::PublicationBuiltinTopicData::durability

durability policy of the corresponding **DataWriter** (p. 499)

6.136.2.6 DurabilityServiceQosPolicy DDS::PublicationBuiltinTopicData::durability_service

durability_service policy of the corresponding **DataWriter** (p. 499)

6.136.2.7 DeadlineQosPolicy DDS::PublicationBuiltinTopicData::deadline

Policy of the corresponding **DataWriter** (p. 499).

6.136.2.8 LatencyBudgetQosPolicy DDS::PublicationBuiltinTopicData::latency_budget

Policy of the corresponding **DataWriter** (p. 499).

6.136.2.9 LivelinessQosPolicy DDS::PublicationBuiltinTopicData::liveliness

Policy of the corresponding **DataWriter** (p. 499).

6.136.2.10 ReliabilityQosPolicy DDS::PublicationBuiltinTopicData::reliability

Policy of the corresponding **DataWriter** (p. 499).

6.136.2.11 LifespanQosPolicy DDS::PublicationBuiltinTopicData::lifespan

Policy of the corresponding **DataWriter** (p. 499).

6.136.2.12 OwnershipQosPolicy DDS::PublicationBuiltinTopicData::ownership

Policy of the corresponding **DataWriter** (p. 499).

6.136.2.13 OwnershipStrengthQosPolicy DDS::PublicationBuiltinTopicData::ownership_strength

Policy of the corresponding **DataWriter** (p. 499).

6.136.2.14 DestinationOrderQosPolicy DDS::PublicationBuiltinTopicData::destination_order

Policy of the corresponding **DataWriter** (p. 499).

6.136.2.15 PresentationQosPolicy DDS::PublicationBuiltinTopicData::presentation

Policy of the **Publisher** (p. 1044) to which the **DataWriter** (p. 499) belongs.

6.136.2.16 BuiltinTopicKey_t DDS::PublicationBuiltinTopicData::publisher_key

<<*eXtension*>> *(p. 174)* DCPS key of the publisher to which the **DataWriter** (p. 499) belongs

6.136.2.17 GUID_t DDS::PublicationBuiltinTopicData::virtual_guid

<<*eXtension*>> *(p. 174)* Virtual GUID associated to the **DataWriter** (p. 499).

See also:

DDS::GUID_t (p. 894)

6.136.2.18 ProtocolVersion_t DDS::PublicationBuiltinTopicData::rtps_protocol_version

<<*eXtension*>> *(p. 174)* Version number of the RTPS wire protocol used.

6.136.2.19 VendorId_t DDS::PublicationBuiltinTopicData::rtps_vendor_id

<<*eXtension*>> *(p. 174)* ID of vendor implementing the RTPS wire protocol.

6.136.2.20 ProductVersion_t DDS::PublicationBuiltinTopicData::product_version

<<*eXtension*>> *(p. 174)* This is a vendor specific parameter. It gives the current version for rti-dds.

6.136.2.21 LocatorFilterQosPolicy ^ DDS::PublicationBuiltinTopicData::locator_filter

<<*eXtension*>> *(p. 174)* Policy of the corresponding **DataWriter** (p. 499)

Related to **DDS::MultiChannelQosPolicy** (p. 981).

6.136.3 Property Documentation

6.136.3.1 UserDataQosPolicy^ DDS::PublicationBuiltinTopicData::user_data [get]

Policy of the corresponding **DataWriter** (p. 499).

6.136.3.2 PartitionQosPolicy^ DDS::PublicationBuiltinTopicData::partition [get]

Policy of the **Publisher** (p. 1044) to which the **DataWriter** (p. 499) belongs.

6.136.3.3 TopicDataQosPolicy^ DDS::PublicationBuiltinTopicData::topic_data [get]

Policy of the related **Topic** (p. 1258).

6.136.3.4 GroupDataQosPolicy^ DDS::PublicationBuiltinTopicData::group_data [get]

Policy of the **Publisher** (p. 1044) to which the **DataWriter** (p. 499) belongs.

6.136.3.5 DDS:: TypeCode^ DDS::PublicationBuiltinTopicData::type_code [get]

<<***eXtension***>> *(p. 174)* Type code information of the corresponding **Topic** (p. 1258)

6.136.3.6 PropertyQosPolicy^ DDS::PublicationBuiltinTopicData::property_qos [get]

<<***eXtension***>> *(p. 174)* Properties of the corresponding **DataWriter** (p. 499).

6.136.3.7 LocatorSeq^ DDS::PublicationBuiltinTopicData::unicast_locators [get]

<<*eXtension*>> *(p. 174)* Custom unicast locators that the endpoint can specify. The default locators will be used if this is not specified.

6.136.3.8 System:: Boolean DDS::PublicationBuiltinTopicData::disable_positive_acks [get, set]

<<*eXtension*>> *(p. 174)* This is a vendor specific parameter. Determines whether matching DataReaders send positive acknowledgements for reliability.

6.137 DDS::PublicationBuiltinTopicDataDataReader Class Reference

Instantiates `DataReader` (p. 433) < **DDS::PublicationBuiltinTopicData** (p. 1030) > .

`#include <managed_builtin.h>`

Inheritance diagram for DDS::PublicationBuiltinTopicDataDataReader::

6.137.1 Detailed Description

Instantiates `DataReader` (p. 433) < **DDS::PublicationBuiltinTopicData** (p. 1030) > .

DDS::DataReader (p. 433) of topic **DDS::PublicationBuiltinTopicDataTypeSupport::PUBLIC_TOPIC_NAME** (p. 230) used for accessing **DDS::PublicationBuiltinTopicData** (p. 1030) of the remote **DDS::DataWriter** (p. 499) and the associated **DDS::Publisher** (p. 1044).

Instantiates:

<<*generic*>> *(p. 175)* **DDS::TypedDataReader** (p. 1338)

See also:

DDS::PublicationBuiltinTopicData (p. 1030)
DDS::PublicationBuiltinTopicDataTypeSupport::PUBLICATION_TOPIC_NAME (p. 230)

6.138 DDS::PublicationBuiltinTopicDataSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < **DDS::PublicationBuiltinTopicData** (p. 1030) > .

`#include <managed_builtin.h>`

6.138.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < **DDS::PublicationBuiltinTopicData** (p. 1030) > .

Instantiates:

<<*generic*>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

DDS::PublicationBuiltinTopicData (p. 1030)

6.139 DDS::PublicationBuiltinTopicDataTypeSupport Class Reference

Instantiates `TypeSupport` (p. 1385) < **DDS::PublicationBuiltinTopicData** (p. 1030) > .

`#include <managed_builtin.h>`

Inherits DDS::AbstractBuiltinTopicDataTypeSupport< T >.

Properties

^ static System::String^ **PUBLICATION_TOPIC_NAME** `[get]`

Publication topic name.

6.139.1 Detailed Description

Instantiates `TypeSupport` (p. 1385) < **DDS::PublicationBuiltinTopicData** (p. 1030) > .

Instantiates:

<<***generic***>> *(p. 175)* **FooTypeSupport** (p. 884)

See also:

DDS::PublicationBuiltinTopicData (p. 1030)

6.140 DDS::PublicationMatchedStatus Struct Reference

DDS::StatusKind::PUBLICATION_MATCHED_STATUS

`#include <managed_publication.h>`

Public Attributes

- System::Int32 **total_count**

  *The total cumulative number of times the concerned **DDS::DataWriter** (p. 499) discovered a "match" with a **DDS::DataReader** (p. 433).*

- System::Int32 **total_count_change**

  The incremental changes in total_count since the last time the listener was called or the status was read.

- System::Int32 **current_count**

  *The current number of readers with which the **DDS::DataWriter** (p. 499) is matched.*

- System::Int32 **current_count_peak**

  <<**eXtension**>> (p. 174) *The highest value that current_count has reached until now.*

- System::Int32 **current_count_change**

  The change in current_count since the last time the listener was called or the status was read.

- **InstanceHandle_t last_subscription_handle**

  *A handle to the last **DDS::DataReader** (p. 433) that caused the the **DDS::DataWriter** (p. 499)'s status to change.*

6.140.1 Detailed Description

DDS::StatusKind::PUBLICATION_MATCHED_STATUS

A "match" happens when the **DDS::DataWriter** (p. 499) finds a **DDS::DataReader** (p. 433) for the same **DDS::Topic** (p. 1258) and common partition with a requested QoS that is compatible with that offered by the **DDS::DataWriter** (p. 499).

This status is also changed (and the listener, if any, called) when a match is ended. A local **DDS::DataWriter** (p. 499) will become "unmatched" from a

remote **DDS::DataReader** (p. 433) when that **DDS::DataReader** (p. 433) goes away for any reason.

6.140.2 Member Data Documentation

6.140.2.1 System::Int32 DDS::PublicationMatchedStatus::total_count

The total cumulative number of times the concerned **DDS::DataWriter** (p. 499) discovered a "match" with a **DDS::DataReader** (p. 433).

This number increases whenever a new match is discovered. It does not change when an existing match goes away.

6.140.2.2 System::Int32 DDS::PublicationMatchedStatus::total_count_change

The incremental changes in total_count since the last time the listener was called or the status was read.

6.140.2.3 System::Int32 DDS::PublicationMatchedStatus::current_count

The current number of readers with which the **DDS::DataWriter** (p. 499) is matched.

This number increases when a new match is discovered and decreases when an existing match goes away.

6.140.2.4 System::Int32 DDS::PublicationMatchedStatus::current_count_peak

*<<**eXtension**>>* *(p. 174)* The highest value that current_count has reached until now.

6.140.2.5 System::Int32 DDS::PublicationMatchedStatus::current_count_change

The change in current_count since the last time the listener was called or the status was read.

6.140.2.6 InstanceHandle_t DDS::PublicationMatchedStatus::last_subscription_handle

A handle to the last **DDS::DataReader** (p. 433) that caused the the **DDS::DataWriter** (p. 499)'s status to change.

6.141 DDS::Publisher Class Reference

<<***interface***>> *(p. 175)* A publisher is the object responsible for the actual dissemination of publications.

`#include <managed_publication.h>`

Inheritance diagram for DDS::Publisher::

Public Member Functions

- ^ void **get_default_datawriter_qos** (**DataWriterQos**^ qos)

  *Copies the default **DDS::DataWriterQos** (p. 546) values into the provided **DDS::DataWriterQos** (p. 546) instance.*

- ^ void **set_default_datawriter_qos** (**DataWriterQos**^ qos)

  *Sets the default **DDS::DataWriterQos** (p. 546) values for this publisher.*

- ^ void **set_default_datawriter_qos_with_profile** (System::String^ library_name, System::String^ profile_name)

  <<**eXtension**>> (p. 174) *Set the default **DDS::DataWriterQos** (p. 546) values for this publisher based on the input XML QoS profile.*

- ^ void **set_default_library** (System::String^ library_name)

  <<**eXtension**>> (p. 174) *Sets the default XML library for a **DDS::Publisher** (p. 1044).*

- ^ System::String^ **get_default_library** ()

  <<**eXtension**>> (p. 174) *Gets the default XML library associated with a **DDS::Publisher** (p. 1044).*

- ^ void **set_default_profile** (System::String^ library_name, System::String^ profile_name)

  <<**eXtension**>> (p. 174) *Sets the default XML profile for a **DDS::Publisher** (p. 1044).*

- ^ System::String^ **get_default_profile** ()

  <<**eXtension**>> (p. 174) *Gets the default XML profile associated with a **DDS::Publisher** (p. 1044).*

- ^ System::String^ **get_default_profile_library** ()

  <<**eXtension**>> (p. 174) *Gets the library where the default XML QoS profile is contained for a **DDS::Publisher** (p. 1044).*

^ **DataWriter**^ **create_datawriter** (**Topic**^ topic, **DataWriterQos**^ qos, **DataWriterListener**^ listener, **StatusMask** mask)

*Creates a **DDS::DataWriter** (p. 499) that will be attached and belong to the **DDS::Publisher** (p. 1044).*

^ **DataWriter**^ **create_datawriter_with_profile** (**Topic**^ topic, System::String^ library_name, System::String^ profile_name, **DataWriterListener**^ listener, **StatusMask** mask)

<<**eXtension**>> (p. 174) *Creates a **DDS::DataWriter** (p. 499) object using the **DDS::DataWriterQos** (p. 546) associated with the input XML QoS profile.*

^ void **delete_datawriter** (**DataWriter**^ %a_datawriter)

*Deletes a **DDS::DataWriter** (p. 499) that belongs to the **DDS::Publisher** (p. 1044).*

^ **DataWriter**^ **lookup_datawriter** (System::String^ topic_name)

*Retrieves the **DDS::DataWriter** (p. 499) for a specific **DDS::Topic** (p. 1258).*

^ void **suspend_publications** ()

*Indicates to RTI Data Distribution Service that the application is about to make multiple modifications using **DDS::DataWriter** (p. 499) objects belonging to the **DDS::Publisher** (p. 1044).*

^ void **resume_publications** ()

*Indicates to RTI Data Distribution Service that the application has completed the multiple changes initiated by the previous **DDS::Publisher::suspend_publications** (p. 1058).*

^ void **begin_coherent_changes** ()

*Indicates that the application will begin a coherent set of modifications using **DDS::DataWriter** (p. 499) objects attached to the **DDS::Publisher** (p. 1044).*

^ void **end_coherent_changes** ()

*Terminates the coherent set initiated by the matching call to **DDS::Publisher::begin_coherent_changes** (p. 1060).*

^ **DomainParticipant**^ **get_participant** ()

*Returns the **DDS::DomainParticipant** (p. 577) to which the **DDS::Publisher** (p. 1044) belongs.*

^ void **delete_contained_entities** ()

*Deletes all the entities that were created by means of the "create" operation on the **DDS::Publisher** (p. 1044).*

^ void **copy_from_topic_qos** (**DataWriterQos**^ a_datawriter_qos, **TopicQos**^ a_topic_qos)

*Copies the policies in the **DDS::TopicQos** (p. 1280) to the corresponding policies in the **DDS::DataWriterQos** (p. 546).*

^ void **wait_for_acknowledgments** (**Duration_t** max_wait)

*Blocks the calling thread until all data written by reliable **DDS::DataWriter** (p. 499) entities is acknowledged, or until timeout expires.*

^ void **wait_for_asynchronous_publishing** (**Duration_t** max_wait)

<<**eXtension**>> (p. 174) *Blocks the calling thread until asynchronous sending is complete.*

^ void **set_qos** (**PublisherQos**^ qos)

Sets the publisher QoS.

^ void **set_qos_with_profile** (System::String^ library_name, System::String^ profile_name)

<<**eXtension**>> (p. 174) *Change the QoS of this publisher using the input XML QoS profile.*

^ void **get_qos** (**PublisherQos**^ qos)

Gets the publisher QoS.

^ void **set_listener** (**PublisherListener**^ l, **StatusMask** mask)

Sets the publisher listener.

^ **PublisherListener**^ **get_listener** ()

Get the publisher listener.

^ virtual void **enable** () override

*Enables the **DDS::Entity** (p. 845).*

^ virtual **StatusCondition**^ **get_statuscondition** () override

*Allows access to the **DDS::StatusCondition** (p. 1183) associated with the **DDS::Entity** (p. 845).*

^ virtual **StatusMask** **get_status_changes** () override

*Retrieves the list of communication statuses in the **DDS::Entity** (p. 845) that are triggered.*

- virtual **InstanceHandle_t get_instance_handle** () override

  *Allows access to the **DDS::InstanceHandle_t** (p. 905) associated with the **DDS::Entity** (p. 845).*

Properties

- static **DataWriterQos^ DATAWRITER_QOS_DEFAULT** `[get]`

  *Special value for creating **DDS::DataWriter** (p. 499) with default QoS.*

- static **DataWriterQos^ DATAWRITER_QOS_USE_TOPIC_QOS** `[get]`

  *Special value for creating **DDS::DataWriter** (p. 499) with a combination of the default **DDS::DataWriterQos** (p. 546) and the **DDS::TopicQos** (p. 1280).*

6.141.1 Detailed Description

<<***interface***>> (p. 175) A publisher is the object responsible for the actual dissemination of publications.

QoS:

DDS::PublisherQos (p. 1074)

Listener:

DDS::PublisherListener (p. 1069)

A publisher acts on the behalf of one or several **DDS::DataWriter** (p. 499) objects that belong to it. When it is informed of a change to the data associated with one of its **DDS::DataWriter** (p. 499) 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. 1044) and the **DDS::DataWriter** (p. 499).

The following operations may be called even if the **DDS::Publisher** (p. 1044) is not enabled. Other operations will fail with the value **DDS::Retcode_NotEnabled** (p. 1121) if called on a disabled **DDS::Publisher** (p. 1044):

- The base-class operations **DDS::Publisher::set_qos** (p. 1063), **DDS::Publisher::set_qos_with_profile** (p. 1064),

DDS::Publisher::get_qos (p. 1065), **DDS::Publisher::set_listener** (p. 1065), **DDS::Publisher::get_listener** (p. 1066), **DDS::Entity::enable** (p. 848), **DDS::Entity::get_statuscondition** (p. 849), **DDS::Entity::get_status_changes** (p. 850)

^ **DDS::Publisher::create_datawriter** (p. 1053), **DDS::Publisher::create_datawriter_with_profile** (p. 1055), **DDS::Publisher::delete_datawriter** (p. 1056), **DDS::Publisher::delete_contained_entities** (p. 1061), **DDS::Publisher::set_default_datawriter_qos** (p. 1049), **DDS::Publisher::set_default_datawriter_qos_with_profile** (p. 1050), **DDS::Publisher::get_default_datawriter_qos** (p. 1048), **DDS::Publisher::wait_for_acknowledgments** (p. 1062), **DDS::Publisher::set_default_library** (p. 1051), **DDS::Publisher::set_default_profile** (p. 1052),

See also:

Operations Allowed in Listener Callbacks (p. 954)

Examples:

HelloWorld_publisher.cpp.

6.141.2 Member Function Documentation

6.141.2.1 void DDS::Publisher::get_default_datawriter_qos (DataWriterQos^ *qos*)

Copies the default **DDS::DataWriterQos** (p. 546) values into the provided **DDS::DataWriterQos** (p. 546) instance.

The retreived `qos` will match the set of values specified on the last successful call to **DDS::Publisher::set_default_datawriter_qos** (p. 1049) or **DDS::Publisher::set_default_datawriter_qos_with_profile** (p. 1050), or else, if the call was never made, the default values from is owning **DDS::DomainParticipant** (p. 577).

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

MT Safety:

UNSAFE. It is not safe to retrieve the default QoS value from a **DDS::Publisher** (p. 1044) while another thread may be simultaneously calling **DDS::Publisher::set_default_datawriter_qos** (p. 1049).

Parameters:

qos *<<**inout**>>* *(p. 176)* **DDS::DataWriterQos** (p. 546) to be filled-up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::Publisher::DATAWRITER_QOS_DEFAULT (p. 80)
DDS::Publisher::create_datawriter (p. 1053)

6.141.2.2 void DDS::Publisher::set_default_datawriter_qos (DataWriterQos^ *qos*)

Sets the default **DDS::DataWriterQos** (p. 546) values for this publisher.

This call causes the default values inherited from the owning **DDS::DomainParticipant** (p. 577) to be overridden.

This default value will be used for newly created **DDS::DataWriter** (p. 499) if **DDS::Publisher::DATAWRITER_QOS_DEFAULT** (p. 80) is specified as the `qos` parameter when **DDS::Publisher::create_datawriter** (p. 1053) is called.

Precondition:

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **DDS::Retcode_InconsistentPolicy** (p. 1119)

MT Safety:

UNSAFE. It is not safe to set the default QoS value from a **DDS::Publisher** (p. 1044) while another thread may be simultaneously calling **DDS::Publisher::set_default_datawriter_qos** (p. 1049), **DDS::Publisher::get_default_datawriter_qos** (p. 1048) or calling **DDS::Publisher::create_datawriter** (p. 1053) with **DDS::Publisher::DATAWRITER_QOS_DEFAULT** (p. 80) as the `qos` parameter.

Parameters:

qos *<<**in**>>* *(p. 175)* Default qos to be set. The special value **DDS::Subscriber::DATAREADER_QOS_DEFAULT** (p. 95) may be passed as `qos` to indicate that the default QoS should be reset back to the initial values the factory would used if **DDS::Publisher::set_default_datawriter_qos** (p. 1049) had never been called. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_InconsistentPolicy** (p. 1119)

6.141.2.3 void DDS::Publisher::set_default_datawriter_qos_with_profile (System::String^ *library_name*, System::String^ *profile_name*)

<<eXtension>> (p. 174) Set the default **DDS::DataWriterQos** (p. 546) values for this publisher based on the input XML QoS profile.

This default value will be used for newly created **DDS::DataWriter** (p. 499) if **DDS::Publisher::DATAWRITER_QOS_DEFAULT** (p. 80) is specified as the `qos` parameter when **DDS::Publisher::create_datawriter** (p. 1053) is called.

Precondition:

The **DDS::DataWriterQos** (p. 546) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **DDS::Retcode_InconsistentPolicy** (p. 1119)

MT Safety:

UNSAFE. It is not safe to set the default QoS value from a **DDS::Publisher** (p. 1044) while another thread may be simultaneously calling **DDS::Publisher::set_default_datawriter_qos** (p. 1049), **DDS::Publisher::get_default_datawriter_qos** (p. 1048) or calling **DDS::Publisher::create_datawriter** (p. 1053) with **DDS::Publisher::DATAWRITER_QOS_DEFAULT** (p. 80) as the `qos` parameter.

Parameters:

library_name *<<in>> (p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::Publisher::set_default_library** (p. 1051)).

profile_name *<<in>> (p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::Publisher::set_default_profile** (p. 1052)).

If the input profile cannot be found, the method fails with **DDS::Retcode_Error** (p. 1116).

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_InconsistentPolicy** (p. 1119)

See also:

DDS::Publisher::DATAWRITER_QOS_DEFAULT (p. 80)
DDS::Publisher::create_datawriter_with_profile (p. 1055)

6.141.2.4 void DDS::Publisher::set_default_library (System::String^ *library_name*)

*<<**eXtension**>>* *(p. 174)* Sets the default XML library for a **DDS::Publisher** (p. 1044).

This method specifies the library that will be used as the default the next time a default library is needed during a call to one of this Publisher's operations.

Any API requiring a library_name as a parameter can use null to refer to the default library.

If the default library is not set, the **DDS::Publisher** (p. 1044) inherits the default from the **DDS::DomainParticipant** (p. 577) (see **DDS::DomainParticipant::set_default_library** (p. 596)).

Parameters:

library_name *<<**in**>>* *(p. 175)* Library name. If library_name is null any previous default is unset.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::Publisher::get_default_library (p. 1051)

6.141.2.5 System::String ^ DDS::Publisher::get_default_library ()

*<<**eXtension**>>* *(p. 174)* Gets the default XML library associated with a **DDS::Publisher** (p. 1044).

Returns:

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

See also:

DDS::Publisher::set_default_library (p. 1051)

6.141.2.6 void DDS::Publisher::set_default_profile (System::String^ *library_name*, System::String^ *profile_name*)

<<eXtension>> *(p. 174)* Sets the default XML profile for a **DDS::Publisher** (p. 1044).

This method specifies the profile that will be used as the default the next time a default **Publisher** (p. 1044) profile is needed during a call to one of this Publishers operations. When calling a **DDS::Publisher** (p. 1044) method that requires a `profile_name` parameter, you can use NULL to refer to the default profile. (This same information applies to setting a default library.)

If the default profile is not set, the **DDS::Publisher** (p. 1044) inherits the default from the **DDS::DomainParticipant** (p. 577) (see **DDS::DomainParticipant::set_default_profile** (p. 597)).

This method does not set the default QoS for **DDS::DataWriter** (p. 499) objects created by the **DDS::Publisher** (p. 1044); for this functionality, use **DDS::Publisher::set_default_datawriter_qos_with_profile** (p. 1050) (you may pass in NULL after having called **set_default_profile()** (p. 1052)).

This method does not set the default QoS for newly created Publishers; for this functionality, use **DDS::DomainParticipant::set_default_publisher_qos_with_profile** (p. 611).

Parameters:

library_name *<<in>>* *(p. 175)* The library name containing the profile.

profile_name *<<in>>* *(p. 175)* The profile name. If profile_name is null any previous default is unset.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::Publisher::get_default_profile (p. 1052)
DDS::Publisher::get_default_profile_library (p. 1053)

6.141.2.7 System::String ^ DDS::Publisher::get_default_profile ()

<<eXtension>> *(p. 174)* Gets the default XML profile associated with a **DDS::Publisher** (p. 1044).

Returns:

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

See also:

DDS::Publisher::set_default_profile (p. 1052)

6.141.2.8 System::String ^ DDS::Publisher::get_default_profile_library ()

<<eXtension>> *(p. 174)* Gets the library where the default XML QoS profile is contained for a **DDS::Publisher** (p. 1044).

The default profile library is automatically set when **DDS::Publisher::set_default_profile** (p. 1052) is called.

This library can be different than the **DDS::Publisher** (p. 1044) default library (see **DDS::Publisher::get_default_library** (p. 1051)).

Returns:

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

See also:

DDS::Publisher::set_default_profile (p. 1052)

6.141.2.9 DataWriter ^ DDS::Publisher::create_datawriter (Topic^ *topic*, DataWriterQos^ *qos*, DataWriterListener^ *listener*, StatusMask *mask*)

Creates a **DDS::DataWriter** (p. 499) that will be attached and belong to the **DDS::Publisher** (p. 1044).

For each application-defined type, `Foo` (p. 877), there is an implied, auto-generated class `DDS::TypedDataWriter` (p. 1368) that extends **DDS::DataWriter** (p. 499) and contains the operations to write data of type `Foo` (p. 877).

Note that a common application pattern to construct the QoS for the **DDS::DataWriter** (p. 499) is to:

- Retrieve the QoS policies on the associated **DDS::Topic** (p. 1258) by means of the **DDS::Topic::get_qos** (p. 1262) operation.
- Retrieve the default **DDS::DataWriter** (p. 499) qos by means of the **DDS::Publisher::get_default_datawriter_qos** (p. 1048) operation.

- Combine those two QoS policies (for example, using **DDS::Publisher::copy_from_topic_qos** (p. 1061)) and selectively modify policies as desired.

When a **DDS::DataWriter** (p. 499) is created, only those transports already registered are available to the **DDS::DataWriter** (p. 499). See **Built-in Transport Plugins** (p. 122) 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. 499) will be created.
The given **DDS::Topic** (p. 1258) must have been created from the same participant as this publisher. If it was created from a different participant, this method will fail.

MT Safety:

UNSAFE. If **DDS::Publisher::DATAWRITER_QOS_DEFAULT** (p. 80) 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. 1049).

Parameters:

topic *<<in>>* *(p. 175)* The **DDS::Topic** (p. 1258) that the **DDS::DataWriter** (p. 499) will be associated with. Cannot be NULL.

qos *<<in>>* *(p. 175)* QoS to be used for creating the new **DDS::DataWriter** (p. 499). The special value **DDS::Publisher::DATAWRITER_QOS_DEFAULT** (p. 80) can be used to indicate that the **DDS::DataWriter** (p. 499) should be created with the default **DDS::DataWriterQos** (p. 546) set in the **DDS::Publisher** (p. 1044). The special value DDS::DATAWRITER_QOS_USE_TOPIC_QOS can be used to indicate that the **DDS::DataWriter** (p. 499) should be created with the combination of the default **DDS::DataWriterQos** (p. 546) set on the **DDS::Publisher** (p. 1044) and the **DDS::TopicQos** (p. 1280) of the **DDS::Topic** (p. 1258). Cannot be NULL.

listener *<<in>>* *(p. 175)* The listener of the **DDS::DataWriter** (p. 499).

mask *<<in>>* *(p. 175)*. Changes of communication status to be invoked on the listener.

Returns:

A **DDS::DataWriter** (p. 499) of a derived class specific to the data type associated with the **DDS::Topic** (p. 1258) or NULL if an error occurred.

See also:

DDS::TypedDataWriter (p. 1368)
Specifying QoS on entities (p. 267) for information on setting QoS before entity creation
DDS::DataWriterQos (p. 546) for rules on consistency among QoS
DDS::Publisher::DATAWRITER_QOS_DEFAULT (p. 80)
DDS::DATAWRITER_QOS_USE_TOPIC_QOS
DDS::Publisher::create_datawriter_with_profile (p. 1055)
DDS::Publisher::get_default_datawriter_qos (p. 1048)
DDS::Topic::set_qos (p. 1261)
DDS::Publisher::copy_from_topic_qos (p. 1061)
DDS::DataWriter::set_listener (p. 515)

Examples:

HelloWorld_publisher.cpp.

6.141.2.10 DataWriter ^ DDS::Publisher::create_datawriter_with_profile (Topic^ *topic*, System::String^ *library_name*, System::String^ *profile_name*, DataWriterListener^ *listener*, StatusMask *mask*)

<<*eXtension*>> *(p. 174)* Creates a **DDS::DataWriter** (p. 499) object using the **DDS::DataWriterQos** (p. 546) associated with the input XML QoS profile.

The **DDS::DataWriter** (p. 499) will be attached and belong to the **DDS::Publisher** (p. 1044).

For each application-defined type, `Foo` (p. 877), there is an implied, auto-generated class `DDS::TypedDataWriter` (p. 1368) that extends **DDS::DataWriter** (p. 499) and contains the operations to write data of type `Foo` (p. 877).

When a **DDS::DataWriter** (p. 499) is created, only those transports already registered are available to the **DDS::DataWriter** (p. 499). See **Built-in Transport Plugins** (p. 122) 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. 499) will be created.
The given **DDS::Topic** (p. 1258) must have been created from the same participant as this publisher. If it was created from a different participant, this method will return NULL.

Parameters:

topic *<<in>>* *(p. 175)* The **DDS::Topic** (p. 1258) that the **DDS::DataWriter** (p. 499) will be associated with. Cannot be NULL.

library_name *<<in>>* *(p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::Publisher::set_default_library** (p. 1051)).

profile_name *<<in>>* *(p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::Publisher::set_default_profile** (p. 1052)).

listener *<<in>>* *(p. 175)* The listener of the **DDS::DataWriter** (p. 499).

mask *<<in>>* *(p. 175)*. Changes of communication status to be invoked on the listener.

Returns:

A **DDS::DataWriter** (p. 499) of a derived class specific to the data type associated with the **DDS::Topic** (p. 1258) or NULL if an error occurred.

See also:

DDS::TypedDataWriter (p. 1368)
Specifying QoS on entities (p. 267) for information on setting QoS before entity creation
DDS::DataWriterQos (p. 546) for rules on consistency among QoS
DDS::Publisher::create_datawriter (p. 1053)
DDS::Publisher::get_default_datawriter_qos (p. 1048)
DDS::Topic::set_qos (p. 1261)
DDS::Publisher::copy_from_topic_qos (p. 1061)
DDS::DataWriter::set_listener (p. 515)

6.141.2.11 void DDS::Publisher::delete_datawriter (DataWriter^ % *a_datawriter*)

Deletes a **DDS::DataWriter** (p. 499) that belongs to the **DDS::Publisher** (p. 1044).

The deletion of the **DDS::DataWriter** (p. 499) will automatically unregister all instances. Depending on the settings of the **WRITER_DATA_LIFECYCLE** (p. 302) QosPolicy, the deletion of the **DDS::DataWriter** (p. 499) may also dispose all instances.

6.141.3 Special Instructions if Using Timestamp APIs and BY_SOURCE_TIMESTAMP Destination Ordering:

If the DataWriters **DDS::DestinationOrderQosPolicy::kind** (p. 562) is DDS::DestinationOrderQosPolicyKind::BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS, calls to **delete_datawriter()** (p. 1056) may fail if your application has previously used the with timestamp APIs (write_w_timestamp(), register_instance_w_timestamp(), unregister_instance_w_timestamp(), or dispose_w_timestamp()) with a timestamp larger (later) than the time at which **delete_datawriter()** (p. 1056) is called. To prevent **delete_datawriter()** (p. 1056) from failing in this situation, either:

- Change the **WRITER_DATA_LIFECYCLE** (p. 302) QosPolicy so that RTI Data Distribution Service will not autodispose unregistered instances (set **DDS::WriterDataLifecycleQosPolicy::autodispose_unregistered_instances** (p. 1432) to false.) or
- Explicitly call unregister_instance_w_timestamp() for all instances modified with the *_w_timestamp() APIs before calling **delete_datawriter()** (p. 1056).

Precondition:

If the **DDS::DataWriter** (p. 499) does not belong to the **DDS::Publisher** (p. 1044), the operation will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123).

Postcondition:

Listener (p. 952) installed on the **DDS::DataWriter** (p. 499) will not be called after this method completes successfully.

Parameters:

a_datawriter *<<in>>* *(p. 175)* The **DDS::DataWriter** (p. 499) to be deleted.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_PreconditionNotMet** (p. 1123).

6.141.3.1 DataWriter ^ DDS::Publisher::lookup_datawriter (System::String^ *topic_name*)

Retrieves the **DDS::DataWriter** (p. 499) for a specific **DDS::Topic** (p. 1258).

This returned **DDS::DataWriter** (p. 499) is either enabled or disabled.

Parameters:

topic_name *<<in>> (p. 175)* Name of the **DDS::Topic** (p. 1258) associated with the **DDS::DataWriter** (p. 499) that is to be looked up. Cannot be NULL.

Returns:

A **DDS::DataWriter** (p. 499) that belongs to the **DDS::Publisher** (p. 1044) attached to the **DDS::Topic** (p. 1258) with `topic_name`. If no such **DDS::DataWriter** (p. 499) exists, this operation returns NULL.

If more than one **DDS::DataWriter** (p. 499) is attached to the **DDS::Publisher** (p. 1044) 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. 499) in one thread while another thread is simultaneously creating or destroying that **DDS::DataWriter** (p. 499).

6.141.3.2 void DDS::Publisher::suspend_publications ()

Indicates to RTI Data Distribution Service that the application is about to make multiple modifications using **DDS::DataWriter** (p. 499) objects belonging to the **DDS::Publisher** (p. 1044).

It is a **hint** to RTI Data Distribution Service 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. 1059) indicating that the set of modifications has completed.

If the **DDS::Publisher** (p. 1044) is deleted before **DDS::Publisher::resume_publications** (p. 1059) is called, any suspended updates yet to be published will be discarded.

RTI Data Distribution Service 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. 867), DDS::PublishModeQosPolicyKind::ASYNCHRONOUS_PUBLISH_MODE_-QOS **DDS::DataWriter** (p. 499) instances allow the user even finer control of traffic shaping and sample coalescing.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-NotEnabled** (p. 1121).

See also:

DDS::FlowController (p. 867)
DDS::FlowController::trigger_flow (p. 869)
DDS::ON_DEMAND_FLOW_CONTROLLER_NAME
DDS::PublishModeQosPolicy (p. 1077)

6.141.3.3 void DDS::Publisher::resume_publications ()

Indicates to RTI Data Distribution Service that the application has completed the multiple changes initiated by the previous **DDS::Publisher::suspend_-publications** (p. 1058).

This is a **hint** to RTI Data Distribution Service that can be used for example, to batch all the modifications made since the **DDS::Publisher::suspend_-publications** (p. 1058).

RTI Data Distribution Service 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. 867), DDS::PublishModeQosPolicyKind::ASYNCHRONOUS_PUBLISH_MODE_-QOS **DDS::DataWriter** (p. 499) instances allow the user even finer control of traffic shaping and sample coalescing.

Precondition:

A call to **DDS::Publisher::resume_publications** (p. 1059) must match a previous call to **DDS::Publisher::suspend_publications** (p. 1058). Otherwise the operation will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123).

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-PreconditionNotMet** (p. 1123) or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::FlowController (p. 867)
DDS::FlowController::trigger_flow (p. 869)
DDS::ON_DEMAND_FLOW_CONTROLLER_NAME
DDS::PublishModeQosPolicy (p. 1077)

6.141.3.4 void DDS::Publisher::begin_coherent_changes ()

Indicates that the application will begin a coherent set of modifications using **DDS::DataWriter** (p. 499) objects attached to the **DDS::Publisher** (p. 1044).

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. 1044) or one of its **DDS::Subscriber** (p. 1201) s may change, a late-joining **DDS::DataReader** (p. 433) 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. 1060).

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).

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_NotEnabled** (p. 1121).

6.141.3.5 void DDS::Publisher::end_coherent_changes ()

Terminates the coherent set initiated by the matching call to **DDS::Publisher::begin_coherent_changes** (p. 1060).

Precondition:

If there is no matching call to **DDS::Publisher::begin_coherent_changes** (p. 1060) the operation will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123).

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_PreconditionNotMet** (p. 1123) or **DDS::Retcode_NotEnabled** (p. 1121).

6.141.3.6 DomainParticipant ^ DDS::Publisher::get_participant ()

Returns the **DDS::DomainParticipant** (p. 577) to which the **DDS::Publisher** (p. 1044) belongs.

Returns:

the **DDS::DomainParticipant** (p. 577) to which the **DDS::Publisher** (p. 1044) belongs.

6.141.3.7 void DDS::Publisher::delete_contained_entities ()

Deletes all the entities that were created by means of the "create" operation on the **DDS::Publisher** (p. 1044).

Deletes all contained **DDS::DataWriter** (p. 499) objects. Once **DDS::Publisher::delete_contained_entities** (p. 1061) completes successfully, the application may delete the **DDS::Publisher** (p. 1044), knowing that it has no contained **DDS::DataWriter** (p. 499) objects.

The operation will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123) if any of the contained entities is in a state where it cannot be deleted.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_PreconditionNotMet** (p. 1123).

6.141.3.8 void DDS::Publisher::copy_from_topic_qos (DataWriterQos^ *a_datawriter_qos*, TopicQos^ *a_topic_qos*)

Copies the policies in the **DDS::TopicQos** (p. 1280) to the corresponding policies in the **DDS::DataWriterQos** (p. 546).

Copies the policies in the **DDS::TopicQos** (p. 1280) to the corresponding policies in the **DDS::DataWriterQos** (p. 546) (replacing values in the **DDS::DataWriterQos** (p. 546), if present).

This is a "convenience" operation most useful in combination with the operations **DDS::Publisher::get_default_datawriter_qos** (p. 1048) and **DDS::Topic::get_qos** (p. 1262). The operation **DDS::Publisher::copy_from_topic_qos** (p. 1061) can be used to merge the **DDS::DataWriter** (p. 499) default QoS policies with the corresponding ones on the **DDS::Topic** (p. 1258). The resulting QoS can then be used to create a new **DDS::DataWriter** (p. 499), or set its QoS.

This operation does not check the resulting **DDS::DataWriterQos** (p. 546) for consistency. This is because the 'merged' **DDS::DataWriterQos** (p. 546) may not be the final one, as the application can still modify some policies prior to applying the policies to the **DDS::DataWriter** (p. 499).

Parameters:

a_datawriter_qos *<<inout>>* *(p. 176)* **DDS::DataWriterQos** (p. 546) to be filled-up. Cannot be NULL.

a_topic_qos *<<in>>* *(p. 175)* **DDS::TopicQos** (p. 1280) to be merged with **DDS::DataWriterQos** (p. 546). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.141.3.9 void DDS::Publisher::wait_for_acknowledgments (Duration_t *max_wait*)

Blocks the calling thread until all data written by reliable **DDS::DataWriter** (p. 499) entities is acknowledged, or until timeout expires.

This operation blocks the calling thread until either all data written by the reliable **DDS::DataWriter** (p. 499) entities is acknowledged by all matched reliable **DDS::DataReader** (p. 433) entities, or else 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; a return value of TIMEOUT indicates that max_wait elapsed before all the data was acknowledged.

If none of the **DDS::DataWriter** (p. 499) instances have **DDS::ReliabilityQosPolicy** (p. 1094) kind set to RELIABLE, the operation will complete successfully.

Parameters:

max_wait *<<in>>* *(p. 175)* Specifies maximum time to wait for acknowledgements **DDS::Duration_t** (p. 717) .

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_NotEnabled** (p. 1121), **DDS::Retcode_Timeout** (p. 1124)

6.141.3.10 void DDS::Publisher::wait_for_asynchronous_publishing (Duration_t *max_wait*)

*<<**eXtension**>> (p. 174)* 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. 499) entities is sent and acknowledged (if reliable) by all matched **DDS::DataReader** (p. 433) 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. 433) is complete in addition to what **DDS::Publisher::wait_for_acknowledgments** (p. 1062) provides.

If none of the **DDS::DataWriter** (p. 499) instances have **DDS::PublishModeQosPolicy::kind** (p. 1079) set to DDS::PublishModeQosPolicyKind::ASYNCHRONOUS_PUBLISH_MODE_QOS, the operation will complete immediately, with DDS::Exception::RETCODE_OK.

Parameters:

max_wait *<<**in**>> (p. 175)* Specifies maximum time to wait for acknowledgements **DDS::Duration_t** (p. 717).

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_NotEnabled** (p. 1121), **DDS::Retcode_Timeout** (p. 1124)

6.141.3.11 void DDS::Publisher::set_qos (PublisherQos^ *qos*)

Sets the publisher QoS.

This operation modifies the QoS of the **DDS::Publisher** (p. 1044).

The **DDS::PublisherQos::group_data** (p. 1075), **DDS::PublisherQos::partition** (p. 1075) and **DDS::PublisherQos::entity_factory** (p. 1075) can be changed. The other policies are immutable.

Parameters:

qos *<<in>>* *(p. 175)* **DDS::PublisherQos** (p. 1074) to be set to. Policies must be consistent. Immutable policies cannot be changed after **DDS::Publisher** (p. 1044) is enabled. The special value **DDS::DomainParticipant::PUBLISHER_QOS_DEFAULT** (p. 38) can be used to indicate that the QoS of the **DDS::Publisher** (p. 1044) should be changed to match the current default **DDS::PublisherQos** (p. 1074) set in the **DDS::DomainParticipant** (p. 577). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_ImmutablePolicy** (p. 1118), or **DDS::Retcode_InconsistentPolicy** (p. 1119).

See also:

DDS::PublisherQos (p. 1074) for rules on consistency among QoS
set_qos (abstract) (p. 846)
Operations Allowed in Listener Callbacks (p. 954)

6.141.3.12 void DDS::Publisher::set_qos_with_profile (System::String^ *library_name*, System::String^ *profile_name*)

<<eXtension>> *(p. 174)* Change the QoS of this publisher using the input XML QoS profile.

This operation modifies the QoS of the **DDS::Publisher** (p. 1044).

The **DDS::PublisherQos::group_data** (p. 1075), **DDS::PublisherQos::partition** (p. 1075) and **DDS::PublisherQos::entity_factory** (p. 1075) can be changed. The other policies are immutable.

Parameters:

library_name *<<in>>* *(p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::Publisher::set_default_library** (p. 1051)).

profile_name *<<in>>* *(p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::Publisher::set_default_profile** (p. 1052)).

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_-ImmutablePolicy** (p. 1118), or **DDS::Retcode_-InconsistentPolicy** (p. 1119).

See also:

DDS::PublisherQos (p. 1074) for rules on consistency among QoS
Operations Allowed in Listener Callbacks (p. 954)

6.141.3.13 void DDS::Publisher::get_qos (PublisherQos^ *qos*)

Gets the publisher QoS.

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

Parameters:

qos <<*in*>> *(p. 175)* **DDS::PublisherQos** (p. 1074) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

get_qos (abstract) (p. 847)

6.141.3.14 void DDS::Publisher::set_listener (PublisherListener^ *l*, StatusMask *mask*)

Sets the publisher listener.

Parameters:

l <<*in*>> *(p. 175)* **DDS::PublisherListener** (p. 1069) to set to.

mask <<*in*>> *(p. 175)* DDS::StatusMask associated with the **DDS::PublisherListener** (p. 1069).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

set_listener (abstract) (p. 847)

6.141.3.15 PublisherListener ^ DDS::Publisher::get_listener ()

Get the publisher listener.

Returns:

DDS::PublisherListener (p. 1069) of the **DDS::Publisher** (p. 1044).

See also:

get_listener (abstract) (p. 848)

6.141.3.16 virtual void DDS::Publisher::enable () [override, virtual]

Enables the **DDS::Entity** (p. 845).

This operation enables the **Entity** (p. 845). **Entity** (p. 845) objects can be created either enabled or disabled. This is controlled by the value of the **ENTITY_FACTORY** (p. 304) QoS policy on the corresponding factory for the **DDS::Entity** (p. 845).

By default, **ENTITY_FACTORY** (p. 304) is set so that it is not necessary to explicitly call **DDS::Entity::enable** (p. 848) on newly created entities.

The **DDS::Entity::enable** (p. 848) operation is idempotent. Calling enable on an already enabled **Entity** (p. 845) returns OK and has no effect.

If a **DDS::Entity** (p. 845) 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. 849)
- 'factory' operations
- **DDS::Entity::get_status_changes** (p. 850) 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_NotEnabled** (p. 1121).

It is legal to delete an **DDS::Entity** (p. 845) that has not been enabled by calling the proper operation on its factory.

Entities created from a factory that is disabled are created disabled, regardless of the setting of the **DDS::EntityFactoryQosPolicy** (p. 851).

Calling enable on an **Entity** (p. 845) whose factory is not enabled will fail and return **DDS::Retcode_PreconditionNotMet** (p. 1123).

If **DDS::EntityFactoryQosPolicy::autoenable_created_entities** (p. 852) is TRUE, the enable operation on a factory will automatically enable all entities created from that factory.

Listeners associated with an entity are not called until the entity is enabled.

Conditions associated with a disabled entity are "inactive," that is, they have a `trigger_value` == FALSE.

Exceptions:

One of the **Standard Return Codes** (p. 235), **Standard Return Codes** (p. 235) or **DDS::Retcode_PreconditionNotMet** (p. 1123).

Implements **DDS::Entity** (p. 848).

6.141.3.17 virtual StatusCondition ^ DDS::Publisher::get_statuscondition () [override, virtual]

Allows access to the **DDS::StatusCondition** (p. 1183) associated with the **DDS::Entity** (p. 845).

The returned condition can then be added to a **DDS::WaitSet** (p. 1411) so that the application can wait for specific status changes that affect the **DDS::Entity** (p. 845).

Returns:

the status condition associated with this entity.

Implements **DDS::Entity** (p. 849).

6.141.3.18 virtual StatusMask DDS::Publisher::get_status_changes () [override, virtual]

Retrieves the list of communication statuses in the **DDS::Entity** (p. 845) that are triggered.

That is, the list of statuses whose value has changed since the last time the application read the status using the get_*_status() method.

When the entity is first created or if the entity is not enabled, all communication statuses are in the "untriggered" state so the list returned by the get_status_changes operation will be empty.

The list of statuses returned by the get_status_changes operation refers to the status that are triggered on the **Entity** (p. 845) itself and does not include statuses that apply to contained entities.

Returns:

list of communication statuses in the **DDS::Entity** (p. 845) that are triggered.

See also:

Status Kinds (p. 238)

Implements **DDS::Entity** (p. 850).

6.141.3.19 virtual InstanceHandle_t DDS::Publisher::get_instance_handle () [override, virtual]

Allows access to the **DDS::InstanceHandle_t** (p. 905) associated with the **DDS::Entity** (p. 845).

This operation returns the **DDS::InstanceHandle_t** (p. 905) that represents the **DDS::Entity** (p. 845).

Returns:

the instance handle associated with this entity.

Implements **DDS::Entity** (p. 850).

6.142 DDS::PublisherListener Class Reference

<<*interface*>> *(p. 175)* **DDS::Listener** (p. 952) for **DDS::Publisher** (p. 1044) status.

`#include <managed_publication.h>`

Inheritance diagram for DDS::PublisherListener::

Public Member Functions

- virtual void **on_offered_deadline_missed** (**DataWriter**^ writer, **OfferedDeadlineMissedStatus**% status)

  Handles the DDS::StatusKind::OFFERED_DEADLINE_MISSED_STATUS status.

- virtual void **on_liveliness_lost** (**DataWriter**^ writer, **LivelinessLostStatus**% status)

  Handles the DDS::StatusKind::LIVELINESS_LOST_STATUS status.

- virtual void **on_offered_incompatible_qos** (**DataWriter**^ writer, **OfferedIncompatibleQosStatus**^ status)

  Handles the DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS status.

- virtual void **on_publication_matched** (**DataWriter**^ writer, **PublicationMatchedStatus**% status)

  Handles the DDS::StatusKind::PUBLICATION_MATCHED_STATUS status.

- virtual void **on_reliable_writer_cache_changed** (**DataWriter**^ writer, **ReliableWriterCacheChangedStatus**% status)

  <<**eXtension**>> (p. 174) *A change has occurred in the writer's cache of unacknowledged samples.*

- virtual void **on_reliable_reader_activity_changed** (**DataWriter**^ writer, **ReliableReaderActivityChangedStatus**% status)

  <<**eXtension**>> (p. 174) *A matched reliable reader has become active or become inactive.*

- virtual void **on_instance_replaced** (**DataWriter**^ writer, **InstanceHandle_t**% handle)

  *Notifies when an instance is replaced in **DataWriter** (p. 499) queue.*

6.142.1 Detailed Description

<<*interface*>> *(p. 175)* **DDS::Listener** (p. 952) for **DDS::Publisher** (p. 1044) status.

Entity:

DDS::Publisher (p. 1044)

Status:

DDS::StatusKind::LIVELINESS_LOST_STATUS,
DDS::LivelinessLostStatus (p. 958);
DDS::StatusKind::OFFERED_DEADLINE_MISSED_STATUS,
DDS::OfferedDeadlineMissedStatus (p. 989);
DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS,
DDS::OfferedIncompatibleQosStatus (p. 991);
DDS::StatusKind::PUBLICATION_MATCHED_STATUS,
DDS::PublicationMatchedStatus (p. 1041);
DDS::StatusKind::RELIABLE_READER_ACTIVITY_CHANGED_STATUS, **DDS::ReliableReaderActivityChangedStatus** (p. 1098);
DDS::StatusKind::RELIABLE_WRITER_CACHE_CHANGED_STATUS,
DDS::ReliableWriterCacheChangedStatus (p. 1101)

See also:

DDS::Listener (p. 952)
Status Kinds (p. 238)
Operations Allowed in Listener Callbacks (p. 954)

6.142.2 Member Function Documentation

6.142.2.1 virtual void DDS::PublisherListener::on_offered_deadline_missed (DataWriter^ *writer*, OfferedDeadlineMissedStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::OFFERED_DEADLINE_MISSED_STATUS status.

This callback is called when the deadline that the **DDS::DataWriter** (p. 499) has committed through its **DEADLINE** (p. 281) qos policy was not respected for a specific instance. This callback is called for each deadline period elapsed during which the **DDS::DataWriter** (p. 499) failed to provide data for an instance.

Parameters:

writer <<*out*>> *(p. 176)* Locally created **DDS::DataWriter** (p. 499) that triggers the listener callback

status <<*out*>> *(p. 176)* Current deadline missed status of locally created **DDS::DataWriter** (p. 499)

Reimplemented from **DDS::DataWriterListener** (p. 525).

Reimplemented in **DDS::DomainParticipantListener** (p. 678).

6.142.2.2 virtual void DDS::PublisherListener::on_liveliness_lost (DataWriter^ *writer*, LivelinessLostStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::LIVELINESS_LOST_STATUS status.

This callback is called when the liveliness that the **DDS::DataWriter** (p. 499) has committed through its **LIVELINESS** (p. 286) qos policy was not respected; this **DDS::DataReader** (p. 433) entities will consider the **DDS::DataWriter** (p. 499) as no longer "alive/active". This callback will not be called when an already not alive **DDS::DataWriter** (p. 499) simply renames not alive for another liveliness period.

Parameters:

writer <<*out*>> *(p. 176)* Locally created **DDS::DataWriter** (p. 499) that triggers the listener callback

status <<*out*>> *(p. 176)* Current liveliness lost status of locally created **DDS::DataWriter** (p. 499)

Reimplemented from **DDS::DataWriterListener** (p. 526).

Reimplemented in **DDS::DomainParticipantListener** (p. 679).

6.142.2.3 virtual void DDS::PublisherListener::on_offered_incompatible_qos (DataWriter^ *writer*, OfferedIncompatibleQosStatus^ *status*) [inline, virtual]

Handles the DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS status.

This callback is called when the **DDS::DataWriterQos** (p. 546) of the **DDS::DataWriter** (p. 499) was incompatible with what was requested by a **DDS::DataReader** (p. 433). This callback is called when a **DDS::DataWriter** (p. 499) has discovered a **DDS::DataReader** (p. 433) for

the same **DDS::Topic** (p. 1258) and common partition, but with a requested QoS that is incompatible with that offered by the **DDS::DataWriter** (p. 499).

Parameters:

writer <<*out*>> *(p. 176)* Locally created **DDS::DataWriter** (p. 499) that triggers the listener callback

status <<*out*>> *(p. 176)* Current incompatible qos status of locally created **DDS::DataWriter** (p. 499)

Reimplemented from **DDS::DataWriterListener** (p. 526).

Reimplemented in **DDS::DomainParticipantListener** (p. 679).

6.142.2.4 virtual void DDS::PublisherListener::on_publication_matched (DataWriter^ *writer*, PublicationMatchedStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::PUBLICATION_MATCHED_STATUS status.

This callback is called when the **DDS::DataWriter** (p. 499) has found a **DDS::DataReader** (p. 433) that matches the **DDS::Topic** (p. 1258), has a common partition and compatible QoS, or has ceased to be matched with a **DDS::DataReader** (p. 433) that was previously considered to be matched.

Parameters:

writer <<*out*>> *(p. 176)* Locally created **DDS::DataWriter** (p. 499) that triggers the listener callback

status <<*out*>> *(p. 176)* Current publication match status of locally created **DDS::DataWriter** (p. 499)

Reimplemented from **DDS::DataWriterListener** (p. 527).

Reimplemented in **DDS::DomainParticipantListener** (p. 680).

6.142.2.5 virtual void DDS::PublisherListener::on_reliable_writer_cache_changed (DataWriter^ *writer*, ReliableWriterCacheChangedStatus% *status*) [inline, virtual]

<<*eXtension*>> *(p. 174)* A change has occurred in the writer's cache of unacknowledged samples.

Parameters:

writer <<*out*>> *(p. 176)* Locally created **DDS::DataWriter** (p. 499) that triggers the listener callback

status *<<**out**>>* *(p. 176)* Current reliable writer cache changed status of locally created **DDS::DataWriter** (p. 499)

Reimplemented from **DDS::DataWriterListener** (p. 527).

Reimplemented in **DDS::DomainParticipantListener** (p. 680).

6.142.2.6 virtual void DDS::PublisherListener::on_reliable_reader_activity_changed (DataWriter^ *writer*, ReliableReaderActivityChangedStatus% *status*) [inline, virtual]

*<<**eXtension**>>* *(p. 174)* A matched reliable reader has become active or become inactive.

Parameters:

writer *<<**out**>>* *(p. 176)* Locally created **DDS::DataWriter** (p. 499) that triggers the listener callback

status *<<**out**>>* *(p. 176)* Current reliable reader activity changed status of locally created **DDS::DataWriter** (p. 499)

Reimplemented from **DDS::DataWriterListener** (p. 528).

Reimplemented in **DDS::DomainParticipantListener** (p. 680).

6.142.2.7 virtual void DDS::PublisherListener::on_instance_replaced (DataWriter^ *writer*, InstanceHandle_t% *handle*) [inline, virtual]

Notifies when an instance is replaced in **DataWriter** (p. 499) queue.

This callback is called when an instance is replaced by the **DDS::DataWriter** (p. 499) 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.

Parameters:

writer *<<**out**>>* *(p. 176)* Locally created **DDS::DataWriter** (p. 499) that triggers the listener callback

handle *<<**out**>>* *(p. 176)* Handle of the replaced instance

Reimplemented from **DDS::DataWriterListener** (p. 528).

6.143 DDS::PublisherQos Class Reference

QoS policies supported by a **DDS::Publisher** (p. 1044) entity.

`#include <managed_publication.h>`

Public Attributes

- **PresentationQosPolicy presentation**
  Presentation policy, ***PRESENTATION*** *(p. 279).*
- **PartitionQosPolicy^ partition**
  Partition policy, ***PARTITION*** *(p. 289).*
- **GroupDataQosPolicy^ group_data**
  Group data policy, ***GROUP_DATA*** *(p. 275).*
- **EntityFactoryQosPolicy entity_factory**
  Entity *(p. 845) factory policy,* ***ENTITY_FACTORY*** *(p. 304).*
- **AsynchronousPublisherQosPolicy^ asynchronous_publisher**
  <<**eXtension**>> (p. 174) *Asynchronous publishing settings for the* ***DDS::Publisher*** *(p. 1044) and all entities that are created by it.*
- **ExclusiveAreaQosPolicy exclusive_area**
  <<**eXtension**>> (p. 174) *Exclusive area for the* ***DDS::Publisher*** *(p. 1044) and all entities that are created by it.*

6.143.1 Detailed Description

QoS policies supported by a **DDS::Publisher** (p. 1044) entity.

You must set certain members in a consistent manner:

length of DDS::PublisherQos::group_data.value <= **DDS::DomainParticipantQos::resource_limits** (p. 686) .publisher_group_data_max_length

length of DDS::PublisherQos::partition.name <= **DDS::DomainParticipantQos::resource_limits** (p. 686) .max_partitions

combined number of characters (including terminating 0) in DDS::PublisherQos::partition.name <= **DDS::DomainParticipantQos::resource_limits** (p. 686) .max_partition_cumulative_characters

If any of the above are not true, **DDS::Publisher::set_qos** (p. 1063) and **DDS::Publisher::set_qos_with_profile** (p. 1064) will fail with **DDS::Retcode_InconsistentPolicy** (p. 1119) and **DDS::DomainParticipant::create_publisher** (p. 615) will return NULL.

6.143.2 Member Data Documentation

6.143.2.1 PresentationQosPolicy DDS::PublisherQos::presentation

Presentation policy, **PRESENTATION** (p. 279).

6.143.2.2 PartitionQosPolicy ^ DDS::PublisherQos::partition

Partition policy, **PARTITION** (p. 289).

6.143.2.3 GroupDataQosPolicy ^ DDS::PublisherQos::group_data

Group data policy, **GROUP_DATA** (p. 275).

6.143.2.4 EntityFactoryQosPolicy DDS::PublisherQos::entity_factory

Entity (p. 845) factory policy, **ENTITY_FACTORY** (p. 304).

6.143.2.5 AsynchronousPublisherQosPolicy ^ DDS::PublisherQos::asynchronous_publisher

<<***eXtension***>> *(p. 174)* Asynchronous publishing settings for the **DDS::Publisher** (p. 1044) and all entities that are created by it.

6.143.2.6 ExclusiveAreaQosPolicy DDS::PublisherQos::exclusive_area

<<***eXtension***>> *(p. 174)* Exclusive area for the **DDS::Publisher** (p. 1044) and all entities that are created by it.

6.144 DDS::PublisherSeq Class Reference

Declares IDL `sequence` < **DDS::Publisher** (p. 1044) > .

`#include <managed_publication.h>`

Inheritance diagram for DDS::PublisherSeq::

6.144.1 Detailed Description

Declares IDL `sequence` < **DDS::Publisher** (p. 1044) > .

See also:

DDS::Sequence (p. 1163)

6.145 DDS::PublishModeQosPolicy Class Reference

Specifies how RTI Data Distribution Service sends application data on the network. This QoS policy can be used to tell RTI Data Distribution Service to use its *own* thread to send data, instead of the user thread.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_publishmode_qos_policy_name** ()

  *Stringified human-readable name for **DDS::PublishModeQosPolicy*** (p. 1077).

Public Attributes

- **PublishModeQosPolicyKind kind**

  Publishing mode.

- System::String^ **flow_controller_name**

  Name of the associated flow controller.

6.145.1 Detailed Description

Specifies how RTI Data Distribution Service sends application data on the network. This QoS policy can be used to tell RTI Data Distribution Service to use its *own* thread to send data, instead of the user thread.

The publishing mode of a **DDS::DataWriter** (p. 499) determines whether data is written synchronously in the context of the user thread when calling **DDS::TypedDataWriter::write** (p. 1376) or asynchronously in the context of a separate thread internal to the middleware.

Each **DDS::Publisher** (p. 1044) spawns a single asynchronous publishing thread (**DDS::AsynchronousPublisherQosPolicy::thread** (p. 373)) to serve all its asynchronous **DDS::DataWriter** (p. 499) instances.

See also:

DDS::AsynchronousPublisherQosPolicy (p. 371)
DDS::HistoryQosPolicy (p. 898)
DDS::FlowController (p. 867)

Entity:

DDS::DataWriter (p. 499)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)

6.145.2 Usage

The fastest way for RTI Data Distribution Service 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. For instance, to send large data reliably, you must use an asynchronous thread.

When data is written asynchronously, a **DDS::FlowController** (p. 867), 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. 499). 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 **DDS::TypedDataWriter::write** (p. 1376) 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 DDS::Transport_Property_t::gather_send_buffer_count_max (each sample requires at least 2-4 gather-send buffers). Performance can be improved by increasing DDS::Transport_Property_t::gather_send_buffer_count_max. 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. 867)). The number of samples that will be queued is determined by the **DDS::HistoryQosPolicy** (p. 898). When using

DDS::HistoryQosPolicyKind::KEEP_LAST_HISTORY_QOS, only the most recent **DDS::HistoryQosPolicy::depth** (p. 901) 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.

6.145.3 Member Data Documentation

6.145.3.1 PublishModeQosPolicyKind DDS::PublishModeQosPolicy::kind

Publishing mode.

[**default**] DDS::PublishModeQosPolicyKind::SYNCHRONOUS_PUBLISH_MODE_QOS

6.145.3.2 System::String ^ DDS::PublishModeQosPolicy::flow_controller_name

Name of the associated flow controller.

NULL value or zero-length string refers to DDS::DEFAULT_FLOW_CONTROLLER_NAME.

See also:

DDS::DomainParticipant::create_flowcontroller (p. 630)
DDS::DEFAULT_FLOW_CONTROLLER_NAME
DDS::FIXED_RATE_FLOW_CONTROLLER_NAME
DDS::ON_DEMAND_FLOW_CONTROLLER_NAME

[**default**] DDS::DEFAULT_FLOW_CONTROLLER_NAME

6.146 DDS::QosPolicyCount Struct Reference

Type to hold a counter for a DDS::QosPolicyId_t.

`#include <managed_infrastructure.h>`

Public Attributes

- **QosPolicyId_t policy_id**
  The `QosPolicy` *ID.*
- System::Int32 **count**
  a counter

6.146.1 Detailed Description

Type to hold a counter for a DDS::QosPolicyId_t.

6.146.2 Member Data Documentation

6.146.2.1 QosPolicyId_t DDS::QosPolicyCount::policy_id

The `QosPolicy` ID.

6.146.2.2 System::Int32 DDS::QosPolicyCount::count

a counter

6.147 DDS::QosPolicyCountSeq Class Reference

Declares IDL `sequence` < **DDS::QosPolicyCount** (p. 1080) >.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::QosPolicyCountSeq::

6.147.1 Detailed Description

Declares IDL `sequence` < **DDS::QosPolicyCount** (p. 1080) >.

Instantiates:

<<*generic*>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

DDS::QosPolicyCount (p. 1080)

6.148 DDS::QueryCondition Class Reference

<<*interface*>> *(p. 175)* These are specialised **DDS::ReadCondition** (p. 1084) objects that allow the application to also specify a filter on the locally available data.

`#include <managed_subscription.h>`

Inheritance diagram for DDS::QueryCondition::

Public Member Functions

- ^ System::String^ **get_query_expression** ()

  Retrieves the query expression.

- ^ void **get_query_parameters** (**StringSeq**^ query_parameters)

  Retrieves the query parameters.

- ^ void **set_query_parameters** (**StringSeq**^ query_parameters)

  Sets the query parameters.

- ^ virtual System::Boolean **get_trigger_value** () override

  Retrieve the `trigger_value`.

6.148.1 Detailed Description

<<*interface*>> *(p. 175)* These are specialised **DDS::ReadCondition** (p. 1084) 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. 1084) state filter and a content filter expressed as a `query_expression` and `query_parameters`.

The query (`query_expression`) is similar to an SQL WHERE clause and can be parameterised by arguments that are dynamically changeable by the **set_query_parameters()** (p. 1083) operation.

Two query conditions that have the same `query_expression` will require unique query condition content filters if their `query_paramters` differ. Query conditions that differ only in their state masks will share the same query condition content filter.

Queries and Filters Syntax (p. 184) describes the syntax of `query_expression` and `query_parameters`.

6.148.2 Member Function Documentation

6.148.2.1 System::String ^ DDS::QueryCondition::get_query_expression ()

Retrieves the query expression.

6.148.2.2 void DDS::QueryCondition::get_query_parameters (StringSeq^ *query_parameters*)

Retrieves the query parameters.

Parameters:

query_parameters *<<inout>>* *(p. 176)* the query parameters are returned here.

6.148.2.3 void DDS::QueryCondition::set_query_parameters (StringSeq^ *query_parameters*)

Sets the query parameters.

Parameters:

query_parameters *<<in>>* *(p. 175)* the new query parameters

6.148.2.4 virtual System::Boolean DDS::QueryCondition::get_trigger_value () `[override, virtual]`

Retrieve the `trigger_value`.

Returns:

the trigger value.

Reimplemented from **DDS::ReadCondition** (p. 1085).

6.149 DDS::ReadCondition Class Reference

<<***interface***>> *(p. 175)* Conditions specifically dedicated to read operations and attached to one **DDS::DataReader** (p. 433).

`#include <managed_subscription.h>`

Inheritance diagram for DDS::ReadCondition::

Public Member Functions

- ^ System::UInt32 **get_sample_state_mask** ()

  Retrieves the set of `sample_states` *for the condition.*

- ^ System::UInt32 **get_view_state_mask** ()

  Retrieves the set of `view_states` *for the condition.*

- ^ System::UInt32 **get_instance_state_mask** ()

  Retrieves the set of `instance_states` *for the condition.*

- ^ **DataReader^ get_datareader** ()

  Returns the ***DDS::DataReader*** *(p. 433) associated with the* ***DDS::ReadCondition*** *(p. 1084).*

- ^ virtual System::Boolean **get_trigger_value** () override

  Retrieve the `trigger_value`*.*

6.149.1 Detailed Description

<<***interface***>> *(p. 175)* Conditions specifically dedicated to read operations and attached to one **DDS::DataReader** (p. 433).

DDS::ReadCondition (p. 1084) 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 **DDS::TypedDataReader::read** (p. 1341) and **DDS::TypedDataReader::take** (p. 1342) variants.

This allows RTI Data Distribution Service to enable the condition only when suitable information is available. They are to be used in conjunction with a **WaitSet** (p. 1411) as normal conditions.

More than one **DDS::ReadCondition** (p. 1084) may be attached to the same **DDS::DataReader** (p. 433).

Note: If you are using a **ReadCondition** (p. 1084) simply to detect the presence of new data, consider using a **DDS::StatusCondition** (p. 1183) with the DATA_AVAILABLE_STATUS instead, which will perform better in this situation.

6.149.2 Member Function Documentation

6.149.2.1 System::UInt32 DDS::ReadCondition::get_sample_state_mask ()

Retrieves the set of `sample_states` for the condition.

6.149.2.2 System::UInt32 DDS::ReadCondition::get_view_state_mask ()

Retrieves the set of `view_states` for the condition.

6.149.2.3 System::UInt32 DDS::ReadCondition::get_instance_state_mask ()

Retrieves the set of `instance_states` for the condition.

6.149.2.4 DataReader ^ DDS::ReadCondition::get_datareader ()

Returns the **DDS::DataReader** (p. 433) associated with the **DDS::ReadCondition** (p. 1084).

There is exactly one **DDS::DataReader** (p. 433) assicated with each **DDS::ReadCondition** (p. 1084).

Returns:

DDS::DataReader (p. 433) associated with the **DDS::ReadCondition** (p. 1084).

6.149.2.5 virtual System::Boolean DDS::ReadCondition::get_trigger_value () `[override, virtual]`

Retrieve the `trigger_value`.

Returns:

the trigger value.

Implements **DDS::Condition** (p. 408).

Reimplemented in **DDS::QueryCondition** (p. 1083).

6.150 DDS::ReaderDataLifecycleQosPolicy Struct Reference

Controls how a **DataReader** (p. 433) manages the lifecycle of the data that it has received.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_readerdatalifecycle_qos_policy_name** ()

  Stringified human-readable name for ***DDS::ReaderDataLifecycleQosPolicy*** *(p. 1087).*

Public Attributes

- **Duration_t autopurge_nowriter_samples_delay**

  Maximum duration for which the ***DDS::DataReader*** *(p. 433) will maintain information regarding an instance once its* `instance_state` *becomes* ***DDS::InstanceStateKind::NOT_ALIVE_NO_WRITERS_INSTANCE_STATE*** *(p. 909).*

- **Duration_t autopurge_disposed_samples_delay**

  Maximum duration for which the ***DDS::DataReader*** *(p. 433) will maintain samples for an instance once its* `instance_state` *becomes* ***DDS::InstanceStateKind::NOT_ALIVE_DISPOSED_INSTANCE_STATE*** *(p. 909).*

6.150.1 Detailed Description

Controls how a **DataReader** (p. 433) manages the lifecycle of the data that it has received.

When a **DataReader** (p. 433) receives data, it is stored in a receive queue for the **DataReader** (p. 433). The user application may either take the data from the queue or leave it there.

This QoS policy controls whether or not RTI Data Distribution Service 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. 433) must retain information regarding instances that have the instance_state **DDS::InstanceStateKind::NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 909).

Note: This policy is not concerned with keeping reliable reader state or discovery information.

The **DDS::DataReader** (p. 433) 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. 898) and **DDS::ResourceLimitsQosPolicy** (p. 1109).

The **DDS::DataReader** (p. 433) 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. 433) 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. 433) will have taken for that instance will have an `instance_state` of either **DDS::InstanceStateKind::NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 909) or **DDS::InstanceStateKind::NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 909) depending on whether or not the last writer that had ownership of the instance disposed it.

In the absence of **READER_DATA_LIFECYCLE** (p. 303), this behavior could cause problems if the application forgets to take those samples. "Untaken" samples will prevent the **DDS::DataReader** (p. 433) from reclaiming the resources and they would remain in the **DDS::DataReader** (p. 433) indefinitely.

For keyed Topics, the consideration of removing data samples from the receive queue is done on a per instance (key) basis. Thus when RTI Data Distribution Service detects that there are no longer DataWriters alive for a certain key value of a **Topic** (p. 1258) (an instance of the **Topic** (p. 1258)), it can be configured to remove all data samples for that instance (key).

Entity:

DDS::DataReader (p. 433)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **YES** (p. 269)

6.150.2 Member Data Documentation

6.150.2.1 Duration_t DDS::ReaderDataLifecycleQosPolicy::autopurge_nowriter_samples_delay

Maximum duration for which the **DDS::DataReader** (p. 433) will maintain information regarding an instance once its `instance_state` becomes **DDS::InstanceStateKind::NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 909).

After this time elapses, the **DDS::DataReader** (p. 433) will purge all internal information regarding the instance, any "untaken" samples will also be lost.

[default] **DDS::Duration_t::DURATION_INFINITE** (p. 253)

[range] [1 nanosec, 1 year] or **DDS::Duration_t::DURATION_INFINITE** (p. 253)

6.150.2.2 Duration_t DDS::ReaderDataLifecycleQosPolicy::autopurge_disposed_samples_delay

Maximum duration for which the **DDS::DataReader** (p. 433) will maintain samples for an instance once its `instance_state` becomes **DDS::InstanceStateKind::NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 909).

After this time elapses, the **DDS::DataReader** (p. 433) will purge all samples for the instance.

[default] **DDS::Duration_t::DURATION_INFINITE** (p. 253)

[range] [1 nanosec, 1 year] or **DDS::Duration_t::DURATION_INFINITE** (p. 253)

6.151 DDS::ReceiverPoolQosPolicy Class Reference

Configures threads used by RTI Data Distribution Service to receive and process data from transports (for example, UDP sockets).

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_receiverpool_qos_policy_name** ()

  *Stringified human-readable name for **DDS::ReceiverPoolQosPolicy*** (p. 1090)*.*

Public Attributes

- **ThreadSettings_t^ thread**

  Receiver pool thread(s).

- System::Int32 **buffer_size**

  The receive buffer size.

- System::Int32 **buffer_alignment**

  The receive buffer alignment.

6.151.1 Detailed Description

Configures threads used by RTI Data Distribution Service 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. 577)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)

See also:

Controlling CPU Core Affinity for RTI Threads (p. 258)

6.151.2 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 Data Distribution Service to use a separate thread to process the data for a **DDS::DataReader** (p. 433), set a unique port for the **DDS::TransportUnicastQosPolicy** (p. 1296) or **DDS::TransportMulticastQosPolicy** (p. 1287) for the **DDS::DataReader** (p. 433).

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. 577) 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.

6.151.3 Member Data Documentation

6.151.3.1 ThreadSettings_t ^ DDS::ReceiverPoolQosPolicy::thread

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 Solaris: OS default priority

For Linux: OS default priority

For LynxOS: 29

For INTEGRITY: 100

For VxWorks: 71

For all others: OS default priority.

[**default**] The actual value depends on your architecture:

For Windows: OS default stack size

For Solaris: OS default stack size

For Linux: OS default stack size

For LynxOS: 4∗16∗1024

For INTEGRITY: 4∗20∗1024

For VxWorks: 4∗16∗1024

For all others: OS default stack size.

[default] mask DDS::ThreadSettingsKind::THREAD_SETTINGS_FLOATING_POINT | DDS::ThreadSettingsKind::THREAD_SETTINGS_STDIO

6.151.3.2 System::Int32 DDS::ReceiverPoolQosPolicy::buffer_size

The receive buffer size.

The receive buffer is used by the receive thread to store the raw data that arrives over the transport.

In many applications, users will change the configuration of the built-in transport DDS::Transport_Property_t::message_size_max to increase the size of the largest data packet that can be sent or received through the transport. Typically, users will change the UDPv4 transport plugin's DDS::Transport_Property_t::message_size_max to 65536 (64 K), which is the largest packet that can be sent/received via UDP.

Ihe ReceiverPoolQosPolicy's `buffer_size` should be set to be the same value as the maximum DDS::Transport_Property_t::message_size_max across *all* of the transports being used.

If you are using the default configuration of the built-in transports, you should not need to change this buffer size.

In addition, if your application *only* uses transports that support zero-copy, then you do not need to modify the value of buffer_size, even if the DDS::Transport_Property_t::message_size_max of the transport is changed. Transports that support zero-copy do not copy their data into the buffer provided by the receive thread. Instead, they provide the receive thread data in a buffer allocated by the transport itself. The only built-in transport that supports zero-copy is the UDPv4 transport on VxWorks platforms.

[default] 9216

[range] [1, 1 GB]

6.151.3.3 System::Int32 DDS::ReceiverPoolQosPolicy::buffer_alignment

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.

6.152 DDS::ReliabilityQosPolicy Struct Reference

Indicates the level of reliability offered/requested by RTI Data Distribution Service.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_reliability_qos_policy_name** ()

  *Stringified human-readable name for **DDS::ReliabilityQosPolicy** (p. 1094).*

Public Attributes

- **ReliabilityQosPolicyKind kind**

  Kind of reliability.

- **Duration_t max_blocking_time**

  The maximum time a writer may block on a write() call.

6.152.1 Detailed Description

Indicates the level of reliability offered/requested by RTI Data Distribution Service.

Entity:

DDS::Topic (p. 1258), **DDS::DataReader** (p. 433), **DDS::DataWriter** (p. 499)

Status:

DDS::StatusKind::OFFERED_INCOMPATIBLE_QOS_STATUS,
DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS

Properties:

RxO (p. 268) = YES
Changeable (p. 269) = **UNTIL ENABLE** (p. 269)

6.152.2 Usage

This policy indicates the level of reliability requested by a **DDS::DataReader** (p. 433) or offered by a **DDS::DataWriter** (p. 499).

The reliability of a connection between a **DataWriter** (p. 499) and **DataReader** (p. 433) 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 Data Distribution Service will not use any resources to monitor or guarantee that the data sent by a **DataWriter** (p. 499) is received by a **DataReader** (p. 433).

For some use cases, such as the periodic update of sensor values to a GUI displaying the value to a person, DDS::ReliabilityQosPolicyKind::BEST_EFFORT_RELIABILITY_QOS 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** (p. 499) is received reliably by DataReaders. This means that RTI Data Distribution Service must check whether or not data was received, and repair any data that was lost by resending a copy of the data as many times as it takes for the **DataReader** (p. 433) to receive the data. RTI Data Distribution Service uses a reliability protocol configured and tuned by these QoS policies: **DDS::HistoryQosPolicy** (p. 898), **DDS::DataWriterProtocolQosPolicy** (p. 529), **DDS::DataReaderProtocolQosPolicy** (p. 465), and **DDS::ResourceLimitsQosPolicy** (p. 1109).

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 Data Distribution Service is determined by the configuration of the aforementioned QoS policies.

You can configure RTI Data Distribution Service 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::ReliabilityQosPolicyKind::BEST_EFFORT_RELIABILITY_QOS < DDS::ReliabilityQosPolicyKind::RELIABLE_RELIABILITY_QOS. A **DDS::DataWriter** (p. 499) 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. 298) policy. In case the reliability kind is set to DDS::ReliabilityQosPolicyKind::RELIABLE_RELIABILITY_QOS the write operation on the **DDS::DataWriter** (p. 499) 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. 298) to be exceeded. Under these circumstances, the **RELIABILITY** (p. 290) `max_blocking_time` configures the maximum duration the write operation may block.

If the **DDS::ReliabilityQosPolicy::kind** (p. 1097) is set to DDS::ReliabilityQosPolicyKind::RELIABLE_RELIABILITY_QOS, data samples originating from a single **DDS::DataWriter** (p. 499) cannot be made available to the **DDS::DataReader** (p. 433) if there are previous data samples that have not been received yet due to a communication error. In other words, RTI Data Distribution Service will repair the error and resend data samples as needed in order to reconstruct a correct snapshot of the **DDS::DataWriter** (p. 499) history before it is accessible by the **DDS::DataReader** (p. 433).

If the **DDS::ReliabilityQosPolicy::kind** (p. 1097) is set to DDS::ReliabilityQosPolicyKind::BEST_EFFORT_RELIABILITY_QOS, the service will not re-transmit missing data samples. However, for data samples originating from any one **DataWriter** (p. 499) the service will ensure they are stored in the **DDS::DataReader** (p. 433) history in the same order they originated in the **DDS::DataWriter** (p. 499). In other words, the **DDS::DataReader** (p. 433) 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. 898)
DDS::ResourceLimitsQosPolicy (p. 1109)

6.152.3 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. 1097) are considered ordered such that DDS::ReliabilityQosPolicyKind::BEST_EFFORT_RELIABILITY_QOS < DDS::ReliabilityQosPolicyKind::RELIABLE_RELIABILITY_QOS.

6.152.4 Member Data Documentation

6.152.4.1 ReliabilityQosPolicyKind DDS::ReliabilityQosPolicy::kind

Kind of reliability.

[**default**] DDS::ReliabilityQosPolicyKind::BEST_EFFORT_RELIABILITY_QOS for **DDS::DataReader** (p. 433) and **DDS::Topic** (p. 1258), DDS::ReliabilityQosPolicyKind::RELIABLE_RELIABILITY_QOS for **DDS::DataWriter** (p. 499)

6.152.4.2 Duration_t DDS::ReliabilityQosPolicy::max_blocking_time

The maximum time a writer may block on a write() call.

This setting applies only to the case where **DDS::ReliabilityQosPolicy::kind** (p. 1097) = DDS::ReliabilityQosPolicyKind::RELIABLE_RELIABILITY_QOS. **DDS::TypedDataWriter::write** (p. 1376) is allowered to block if the **DDS::DataWriter** (p. 499) does not have space to store the value written. Only applies to **DDS::DataWriter** (p. 499).

[**default**] 100 milliseconds

[**range**] [0,1 year] or **DDS::Duration_t::DURATION_INFINITE** (p. 253)

See also:

DDS::ResourceLimitsQosPolicy (p. 1109)

6.153 DDS::ReliableReaderActivityChangedStatus Struct Reference

<<*eXtension*>> *(p. 174)* Describes the activity (i.e. are acknowledgements forthcoming) of reliable readers matched to a reliable writer.

`#include <managed_publication.h>`

Public Attributes

- System::Int32 **active_count**

  The current number of reliable readers currently matched with this reliable writer.

- System::Int32 **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.

- System::Int32 **active_count_change**

  The most recent change in the number of active remote reliable readers.

- System::Int32 **inactive_count_change**

  The most recent change in the number of inactive remote reliable readers.

- **InstanceHandle_t last_instance_handle**

  The instance handle of the last reliable remote reader to be determined inactive.

6.153.1 Detailed Description

<<*eXtension*>> *(p. 174)* Describes the activity (i.e. are acknowledgements forthcoming) of reliable readers matched to a reliable writer.

Entity:

DDS::DataWriter (p. 499)

Listener:

DDS::DataWriterListener (p. 524)

This status is the reciprocal status to the **DDS::LivelinessChangedStatus** (p. 956) on the reader. It is different than the **DDS::LivelinessLostStatus**

(p. 958) 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.

6.153.2 Member Data Documentation

6.153.2.1 System::Int32 DDS::ReliableReaderActivityChangedStatus::active_count

The current number of reliable readers currently matched with this reliable writer.

6.153.2.2 System::Int32 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. 1133) times, each heartbeat having been separated from the previous by the current heartbeat period.

6.153.2.3 System::Int32 DDS::ReliableReaderActivityChangedStatus::active_count_change

The most recent change in the number of active remote reliable readers.

6.153.2.4 System::Int32 DDS::ReliableReaderActivityChangedStatus::inactive_count_change

The most recent change in the number of inactive remote reliable readers.

6.153.2.5 InstanceHandle_t DDS::ReliableReaderActivityChangedStatus::last_instance_handle

The instance handle of the last reliable remote reader to be determined inactive.

6.154 DDS::ReliableWriterCacheChangedStatus Struct Reference

<<***eXtension***>> *(p. 174)* A summary of the state of a data writer's cache of unacknowledged samples written.

`#include <managed_publication.h>`

Public Attributes

- **ReliableWriterCacheEventCount empty_reliable_writer_cache**

  The number of times the reliable writer's cache of unacknowledged samples has become empty.

- **ReliableWriterCacheEventCount full_reliable_writer_cache**

  The number of times the reliable writer's cache of unacknowledged samples has become full.

- **ReliableWriterCacheEventCount low_watermark_reliable_writer_cache**

  The number of times the reliable writer's cache of unacknowledged samples has fallen to the low watermark.

- **ReliableWriterCacheEventCount high_watermark_reliable_writer_cache**

  The number of times the reliable writer's cache of unacknowledged samples has risen to the high watermark.

- System::Int32 **unacknowledged_sample_count**

  The current number of unacknowledged samples in the writer's cache.

- System::Int32 **unacknowledged_sample_count_peak**

  The highest value that unacknowledged_sample_count has reached until now.

6.154.1 Detailed Description

<<***eXtension***>> *(p. 174)* A summary of the state of a data writer's cache of unacknowledged samples written.

Entity:

DDS::DataWriter (p. 499)

Listener:

DDS::DataWriterListener (p. 524)

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.

6.154.2 Member Data Documentation

6.154.2.1 ReliableWriterCacheEventCount DDS::ReliableWriterCacheChangedStatus::empty_reliable_writer_cache

The number of times the reliable writer's cache of unacknowledged samples has become empty.

6.154.2.2 ReliableWriterCacheEventCount DDS::ReliableWriterCacheChangedStatus::full_reliable_writer_cache

The number of times the reliable writer's cache of unacknowledged samples has become full.

6.154.2.3 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.

6.154.2.4 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.

6.154.2.5 System::Int32 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.

6.154.2.6 System::Int32 DDS::ReliableWriterCacheChangedStatus::unacknowledged_sample_count_peak

The highest value that unacknowledged_sample_count has reached until now.

6.155 DDS::ReliableWriterCacheEventCount Struct Reference

*<<**eXtension**>>* *(p. 174)* The number of times the number of unacknowledged samples in the cache of a reliable writer hit a certain well-defined threshold.

`#include <managed_publication.h>`

Public Attributes

- System::Int32 **total_count**

  The total number of times the event has occurred.

- System::Int32 **total_count_change**

  The incremental number of times the event has occurred since the listener was last invoked or the status read.

6.155.1 Detailed Description

*<<**eXtension**>>* *(p. 174)* The number of times the number of unacknowledged samples in the cache of a reliable writer hit a certain well-defined threshold.

See also:

DDS::ReliableWriterCacheChangedStatus (p. 1101)

6.155.2 Member Data Documentation

6.155.2.1 System::Int32 DDS::ReliableWriterCacheEventCount::total_count

The total number of times the event has occurred.

6.155.2.2 System::Int32 DDS::ReliableWriterCacheEventCount::total_count_change

The incremental number of times the event has occurred since the listener was last invoked or the status read.

6.156 DDS::RequestedDeadlineMissedStatus Struct Reference

DDS::StatusKind::REQUESTED_DEADLINE_MISSED_STATUS

`#include <managed_subscription.h>`

Public Attributes

- System::Int32 **total_count**

  *Total cumulative count of the deadlines detected for any instance read by the **DDS::DataReader** (p. 433).*

- System::Int32 **total_count_change**

  The incremental number of deadlines detected since the last time the listener was called or the status was read.

- **InstanceHandle_t last_instance_handle**

  *Handle to the last instance in the **DDS::DataReader** (p. 433) for which a deadline was detected.*

6.156.1 Detailed Description

DDS::StatusKind::REQUESTED_DEADLINE_MISSED_STATUS

Examples:

HelloWorld_subscriber.cpp.

6.156.2 Member Data Documentation

6.156.2.1 System::Int32 DDS::RequestedDeadlineMissedStatus::total_count

Total cumulative count of the deadlines detected for any instance read by the **DDS::DataReader** (p. 433).

6.156.2.2 System::Int32 DDS::RequestedDeadlineMissedStatus::total_count_change

The incremental number of deadlines detected since the last time the listener was called or the status was read.

6.156.2.3 InstanceHandle_t DDS::RequestedDeadlineMissedStatus::last_instance_handle

Handle to the last instance in the **DDS::DataReader** (p. 433) for which a deadline was detected.

6.157 DDS::RequestedIncompatibleQosStatus Class Reference

DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS

`#include <managed_subscription.h>`

Public Attributes

- System::Int32 **total_count**

  *Total cumulative count of how many times the concerned **DDS::DataReader** (p. 433) discovered a **DDS::DataWriter** (p. 499) for the same **DDS::Topic** (p. 1258) with an offered QoS that is incompatible with that requested by the **DDS::DataReader** (p. 433).*

- System::Int32 **total_count_change**

  The change in `total_count` *since the last time the listener was called or the status was read.*

- **QosPolicyId_t last_policy_id**

  The PolicyId_t of one of the policies that was found to be incompatible the last time an incompatibility was detected.

- **QosPolicyCountSeq^ policies**

  *A list containing, for each policy, the total number of times that the concerned **DDS::DataReader** (p. 433) discovered a **DDS::DataWriter** (p. 499) for the same **DDS::Topic** (p. 1258) with an offered QoS that is incompatible with that requested by the **DDS::DataReader** (p. 433).*

6.157.1 Detailed Description

DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS

See also:

DURABILITY (p. 276)
PRESENTATION (p. 279)
RELIABILITY (p. 290)
OWNERSHIP (p. 283)
LIVELINESS (p. 286)
DEADLINE (p. 281)
LATENCY_BUDGET (p. 282)
DESTINATION_ORDER (p. 292)

Examples:

HelloWorld_subscriber.cpp.

6.157.2 Member Data Documentation

6.157.2.1 System::Int32 DDS::RequestedIncompatibleQosStatus::total_count

Total cumulative count of how many times the concerned **DDS::DataReader** (p. 433) discovered a **DDS::DataWriter** (p. 499) for the same **DDS::Topic** (p. 1258) with an offered QoS that is incompatible with that requested by the **DDS::DataReader** (p. 433).

6.157.2.2 System::Int32 DDS::RequestedIncompatibleQosStatus::total_count_change

The change in `total_count` since the last time the listener was called or the status was read.

6.157.2.3 QosPolicyId_t DDS::RequestedIncompatibleQosStatus::last_policy_id

The PolicyId_t of one of the policies that was found to be incompatible the last time an incompatibility was detected.

6.157.2.4 QosPolicyCountSeq ^ DDS::RequestedIncompatibleQosStatus::policies

A list containing, for each policy, the total number of times that the concerned **DDS::DataReader** (p. 433) discovered a **DDS::DataWriter** (p. 499) for the same **DDS::Topic** (p. 1258) with an offered QoS that is incompatible with that requested by the **DDS::DataReader** (p. 433).

6.158 DDS::ResourceLimitsQosPolicy Struct Reference

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.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- ^ static System::String^ **get_resourcelimits_qos_policy_name** ()

  Stringified human-readable name for ***DDS::ResourceLimitsQosPolicy*** *(p. 1109).*

Public Attributes

- ^ System::Int32 **max_samples**

  Represents the maximum samples the middleware can store for any one ***DDS::DataWriter*** *(p. 499) (or* ***DDS::DataReader*** *(p. 433)).*

- ^ System::Int32 **max_instances**

  Represents the maximum number of instances a ***DDS::DataWriter*** *(p. 499) (or* ***DDS::DataReader*** *(p. 433)) can manage.*

- ^ System::Int32 **max_samples_per_instance**

  Represents the maximum number of samples of any one instance a ***DDS::DataWriter*** *(p. 499) (or* ***DDS::DataReader*** *(p. 433)) can manage.*

- ^ System::Int32 **initial_samples**

  <<**eXtension**>> (p. 174) *Represents the initial samples the middleware will store for any one* ***DDS::DataWriter*** *(p. 499) (or* ***DDS::DataReader*** *(p. 433)).*

- ^ System::Int32 **initial_instances**

  <<**eXtension**>> (p. 174) *Represents the initial number of instances a* ***DDS::DataWriter*** *(p. 499) (or* ***DDS::DataReader*** *(p. 433)) will manage.*

- ^ System::Int32 **instance_hash_buckets**

  <<**eXtension**>> (p. 174) *Number of hash buckets for instances.*

Properties

^ static System::Int32 **LENGTH_UNLIMITED** [get]

A special value indicating an unlimited quantity.

6.158.1 Detailed Description

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. 1258), **DDS::DataReader** (p. 433), **DDS::DataWriter** (p. 499)

Status:

DDS::StatusKind::SAMPLE_REJECTED_STATUS,
DDS::SampleRejectedStatus (p. 1159)

Properties:

RxO (p. 268) = NO
Changeable (p. 269) = **UNTIL ENABLE** (p. 269)

6.158.2 Usage

This policy controls the resources that RTI Data Distribution Service can use to meet the requirements imposed by the application and other QoS settings.

For the reliability protocol (and **DDS::DurabilityQosPolicy** (p. 709)), this QoS policy determines the actual maximum queue size when the **DDS::HistoryQosPolicy** (p. 898) is set to DDS::HistoryQosPolicyKind::KEEP_ALL_HISTORY_QOS.

In general, this QoS policy is used to limit the amount of system memory that RTI Data Distribution Service 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. 499) objects are communicating samples faster than they are ultimately taken by the **DDS::DataReader** (p. 433) 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. 433) cannot keep up with its corresponding **DDS::DataWriter** (p. 499). The behavior in this case depends on the setting for the **RELIABILITY** (p. 290). If reliability is DDS::ReliabilityQosPolicyKind::BEST_EFFORT_RELIABILITY_QOS, then RTI Data Distribution Service is allowed to drop samples. If the reliability is DDS::ReliabilityQosPolicyKind::RELIABLE_RELIABILITY_QOS, RTI Data Distribution Service will block the **DDS::DataWriter** (p. 499) or discard the sample at the **DDS::DataReader** (p. 433) in order not to lose existing samples.

The constant DDS::LENGTH_UNLIMITED may be used to indicate the absence of a particular limit. For example setting **DDS::ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1113) to DDS::LENGTH_UNLIMITED will cause RTI Data Distribution Service not to enforce this particular limit.

If these resource limits are not set sufficiently, under certain circumstances the **DDS::DataWriter** (p. 499) may block on a write() call even though the **DDS::HistoryQosPolicy** (p. 898) is DDS::HistoryQosPolicyKind::KEEP_LAST_HISTORY_QOS. To guarantee the writer does not block for DDS::HistoryQosPolicyKind::KEEP_LAST_HISTORY_QOS, make sure the resource limits are set such that:

```
max_samples >= max_instances * max_samples_per_instance
```

See also:

DDS::ReliabilityQosPolicy (p. 1094)
DDS::HistoryQosPolicy (p. 898)

6.158.3 Consistency

The setting of **DDS::ResourceLimitsQosPolicy::max_samples** (p. 1112) must be consistent with **DDS::ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1113). For these two values to be consistent, it must be true that **DDS::ResourceLimitsQosPolicy::max_samples** (p. 1112) >= **DDS::ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1113). As described above, this limit will not be enforced if **DDS::ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1113) is set to DDS::LENGTH_UNLIMITED.

The setting of **RESOURCE_LIMITS** (p. 298) `max_samples_per_instance` must be consistent with the **HISTORY** (p. 294) `depth`. For these two QoS to be consistent, it must be true that *depth* <= *max_samples_per_instance*.

See also:

DDS::HistoryQosPolicy (p. 898)

6.158.4 Member Data Documentation

6.158.4.1 System::Int32 DDS::ResourceLimitsQosPolicy::max_samples

Represents the maximum samples the middleware can store for any one **DDS::DataWriter** (p. 499) (or **DDS::DataReader** (p. 433)).

Specifies the maximum number of data samples a **DDS::DataWriter** (p. 499) (or **DDS::DataReader** (p. 433)) 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.

When batching is enabled, the maximum number of data samples a **DDS::DataWriter** (p. 499) can manage will also be limited by **DDS::DataWriterResourceLimitsQosPolicy::max_batches** (p. 555).

[default] DDS::LENGTH_UNLIMITED

[range] [1, 100 million] or DDS::LENGTH_UNLIMITED, >= initial_samples, >= max_samples_per_instance, >= **DDS::DataReaderResourceLimitsQosPolicy::max_samples_per_remote_writer** (p. 490) or >= **DDS::RtpsReliableWriterProtocol_t::heartbeats_per_max_samples** (p. 1133)

For **DDS::DataWriterQos** (p. 546) max_samples >= DDS::DataWriterProtocolQosPolicy::rtps_reliable_writer.heartbeats_per_max_samples if batching is disabled.

6.158.4.2 System::Int32 DDS::ResourceLimitsQosPolicy::max_instances

Represents the maximum number of instances a **DDS::DataWriter** (p. 499) (or **DDS::DataReader** (p. 433)) can manage.

[default] DDS::LENGTH_UNLIMITED

[range] [1, 1 million] or DDS::LENGTH_UNLIMITED, >= initial_instances

6.158.4.3 System::Int32 DDS::ResourceLimitsQosPolicy::max_samples_per_instance

Represents the maximum number of samples of any one instance a **DDS::DataWriter** (p. 499) (or **DDS::DataReader** (p. 433)) can manage.

For unkeyed types, this value has to be equal to max_samples or DDS::LENGTH_UNLIMITED.

[**default**] DDS::LENGTH_UNLIMITED

[**range**] [1, 100 million] or DDS::LENGTH_UNLIMITED, <= max_samples or DDS::LENGTH_UNLIMITED, >= **DDS::HistoryQosPolicy::depth** (p. 901)

6.158.4.4 System::Int32 DDS::ResourceLimitsQosPolicy::initial_samples

<<***eXtension***>> *(p. 174)* Represents the initial samples the middleware will store for any one **DDS::DataWriter** (p. 499) (or **DDS::DataReader** (p. 433)).

Specifies the initial number of data samples a **DDS::DataWriter** (p. 499) (or **DDS::DataReader** (p. 433)) will manage across all the instances associated with it.

[**default**] 32

[**range**] [1,100 million], <= max_samples

6.158.4.5 System::Int32 DDS::ResourceLimitsQosPolicy::initial_instances

<<***eXtension***>> *(p. 174)* Represents the initial number of instances a **DDS::DataWriter** (p. 499) (or **DDS::DataReader** (p. 433)) will manage.

[**default**] 32

[**range**] [1,1 million], <= max_instances

6.158.4.6 System::Int32 DDS::ResourceLimitsQosPolicy::instance_hash_buckets

<<***eXtension***>> *(p. 174)* 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]

6.159 DDS::Retcode_AlreadyDeleted Class Reference

The object target of this operation has already been deleted.

`#include <managed_exceptions.h>`

Inheritance diagram for DDS::Retcode_AlreadyDeleted::

6.159.1 Detailed Description

The object target of this operation has already been deleted.

6.160 DDS::Retcode_BadParameter Class Reference

Illegal parameter value.

`#include <managed_exceptions.h>`

Inheritance diagram for DDS::Retcode_BadParameter::

6.160.1 Detailed Description

Illegal parameter value.

The value of the parameter that is passed in has llegal value. Things that falls into this category includes null parameters and parameter values that are out of range.

6.161 DDS::Retcode_Error Class Reference

Generic, unspecified error.

`#include <managed_exceptions.h>`

Inheritance diagram for DDS::Retcode_Error::

6.161.1 Detailed Description

Generic, unspecified error.

6.162 DDS::Retcode_IllegalOperation Class Reference

The operation was called under improper circumstances.

`#include <managed_exceptions.h>`

Inheritance diagram for DDS::Retcode_IllegalOperation::

6.162.1 Detailed Description

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_PreconditionNotMet** (p. 1123), except that there is no precondition that could be changed to make the operation succeed.

6.163 DDS::Retcode_ImmutablePolicy Class Reference

Application attempted to modify an immutable QoS policy.

`#include <managed_exceptions.h>`

Inheritance diagram for DDS::Retcode_ImmutablePolicy::

6.163.1 Detailed Description

Application attempted to modify an immutable QoS policy.

6.164 DDS::Retcode_InconsistentPolicy Class Reference

Application specified a set of QoS policies that are not consistent with each other.

`#include <managed_exceptions.h>`

Inheritance diagram for DDS::Retcode_InconsistentPolicy::

6.164.1 Detailed Description

Application specified a set of QoS policies that are not consistent with each other.

6.165 DDS::Retcode_NoData Class Reference

Indicates a transient situation where the operation did not return any data but there is no inherent error.

`#include <managed_exceptions.h>`

Inheritance diagram for DDS::Retcode_NoData::

6.165.1 Detailed Description

Indicates a transient situation where the operation did not return any data but there is no inherent error.

Examples:

HelloWorld_subscriber.cpp.

6.166 DDS::Retcode_NotEnabled Class Reference

Operation invoked on a **DDS::Entity** (p. 845) that is not yet enabled.

`#include <managed_exceptions.h>`

Inheritance diagram for DDS::Retcode_NotEnabled::

6.166.1 Detailed Description

Operation invoked on a **DDS::Entity** (p. 845) that is not yet enabled.

6.167 DDS::Retcode_OutOfResources Class Reference

RTI Data Distribution Service ran out of the resources needed to complete the operation.

`#include <managed_exceptions.h>`

Inheritance diagram for DDS::Retcode_OutOfResources::

6.167.1 Detailed Description

RTI Data Distribution Service ran out of the resources needed to complete the operation.

6.168 DDS::Retcode_PreconditionNotMet Class Reference

A pre-condition for the operation was not met.

`#include <managed_exceptions.h>`

Inheritance diagram for DDS::Retcode_PreconditionNotMet::

6.168.1 Detailed Description

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.

6.169 DDS::Retcode_Timeout Class Reference

The operation timed out.

`#include <managed_exceptions.h>`

Inheritance diagram for DDS::Retcode_Timeout::

6.169.1 Detailed Description

The operation timed out.

6.170 DDS::Retcode_Unsupported Class Reference

Unsupported operation. Can only returned by operations that are unsupported.

`#include <managed_exceptions.h>`

Inheritance diagram for DDS::Retcode_Unsupported::

6.170.1 Detailed Description

Unsupported operation. Can only returned by operations that are unsupported.

6.171 DDS::RtpsReliableReaderProtocol_t Struct Reference

Qos related to reliable reader protocol defined in RTPS.

`#include <managed_infrastructure.h>`

Public Attributes

- **Duration_t min_heartbeat_response_delay**
  The minimum delay to respond to a heartbeat.
- **Duration_t max_heartbeat_response_delay**
  The maximum delay to respond to a heartbeat.
- **Duration_t heartbeat_suppression_duration**
  The duration a reader ignores consecutively received heartbeats.
- **Duration_t nack_period**
  The period at which to send NACKs.

6.171.1 Detailed Description

Qos related to reliable reader protocol defined in RTPS.

It is used to config reliable reader according to RTPS protocol.

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)

QoS:

DDS::DataReaderProtocolQosPolicy (p. 465)
DDS::DiscoveryConfigQosPolicy (p. 563)

6.171.2 Member Data Documentation

6.171.2.1 Duration_t DDS::RtpsReliableReaderProtocol_t::min_heartbeat_response_delay

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

6.171.2.2 Duration_t DDS::RtpsReliableReaderProtocol_t::max_heartbeat_response_delay

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::DiscoveryConfigQosPolicy** (p. 563) : 0 seconds

For **DDS::DataReaderProtocolQosPolicy** (p. 465) : 0.5 seconds

[**range**] [0, 1 year], >= min_heartbeat_response_delay

6.171.2.3 Duration_t DDS::RtpsReliableReaderProtocol_t::heartbeat_suppression_duration

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],

6.171.2.4 Duration_t DDS::RtpsReliableReaderProtocol_t::nack_period

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]

6.172 DDS::RtpsReliableWriterProtocol_t Struct Reference

QoS related to the reliable writer protocol defined in RTPS.

`#include <managed_infrastructure.h>`

Public Attributes

^ System::Int32 **low_watermark**

When the number of unacknowledged samples in the cache of a reliable writer falls below this threshold, the DDS::StatusKind::RELIABLE_WRITER_-CACHE_CHANGED_STATUS is considered to have changed.

^ System::Int32 **high_watermark**

When the number of unacknowledged samples in the cache of a reliable writer climbs above this threshold, the DDS::StatusKind::RELIABLE_WRITER_-CACHE_CHANGED_STATUS is considered to have changed.

^ **Duration_t heartbeat_period**

The period at which to send heartbeats.

^ **Duration_t fast_heartbeat_period**

An alternative heartbeat period used when a reliable writer needs to flush its unacknowledged samples more quickly.

^ **Duration_t late_joiner_heartbeat_period**

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.

^ System::Int32 **max_heartbeat_retries**

The maximum number of periodic *heartbeat retries before marking a remote reader as inactive.*

^ System::Int32 **heartbeats_per_max_samples**

The number of heartbeats per send queue.

^ **Duration_t min_nack_response_delay**

The minimum delay to respond to a NACK.

^ **Duration_t max_nack_response_delay**

The maximum delay to respond to a nack.

ˆ **Duration_t nack_suppression_duration**

The duration for ignoring consecutive NACKs that may trigger redundant repairs.

ˆ System::Int32 **max_bytes_per_nack_response**

The maximum total message size when resending dropped samples.

ˆ **Duration_t disable_positive_acks_min_sample_keep_duration**

The minimum duration a sample is queued for ACK-disabled readers.

ˆ **Duration_t disable_positive_acks_max_sample_keep_duration**

The maximum duration a sample is queued for ACK-disabled readers.

ˆ System::Int32 **disable_positive_acks_decrease_sample_keep_duration_factor**

Controls rate of contraction of dynamic sample keep duration.

ˆ System::Int32 **disable_positive_acks_increase_sample_keep_duration_factor**

Controls rate of growth of dynamic sample keep duration.

ˆ System::Int32 **min_send_window_size**

Minimum size of send window of unacknowledged samples.

ˆ System::Int32 **max_send_window_size**

Maximum size of send window of unacknowledged samples.

ˆ **Duration_t send_window_update_period**

Period in which send window may be dynamically changed.

ˆ System::Int32 **send_window_increase_factor**

Increases send window size by this percentage when reacting dynamically to network conditions.

ˆ System::Int32 **send_window_decrease_factor**

Decreases send window size by this percentage when reacting dynamically to network conditions.

Properties

^ System::Boolean **inactivate_nonprogressing_readers** `[get, set]`

Whether to treat remote readers as inactive when their NACKs do not progress.

^ System::Boolean **disable_positive_acks_enable_adaptive_sample_keep_duration** `[get, set]`

Enables dynamic adjustment of sample keep duration in response to congestion.

6.172.1 Detailed Description

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. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)

QoS:

DDS::DataWriterProtocolQosPolicy (p. 529)
DDS::DiscoveryConfigQosPolicy (p. 563)

6.172.2 Member Data Documentation

6.172.2.1 System::Int32 DDS::RtpsReliableWriterProtocol_t::low_watermark

When the number of unacknowledged samples in the cache of a reliable writer falls below this threshold, the DDS::StatusKind::RELIABLE_WRITER_CACHE_CHANGED_STATUS is considered to have changed.

This value is measured in units of samples except with batching configurations in non MultiChannel DataWriters 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. 1132) and **DDS::RtpsReliableWriterProtocol_t::fast_heartbeat_period** (p. 1132), respectively). When the number of unacknowledged samples in the queue of a reliable **DDS::DataWriter** (p. 499) exceeds high_watermark, the DDS::StatusKind::RELIABLE_WRITER_CACHE_CHANGED_STATUS is changed, and the **DataWriter** (p. 499) will start heartbeating at fast_heartbeat_rate. When the number of samples falls below low_watermark, DDS::StatusKind::RELIABLE_WRITER_CACHE_CHANGED_STATUS is changed, and the heartbeat rate will return to the "normal" rate (heartbeat_rate).

[**default**] 0

[**range**] [0, 100 million], < high_watermark

See also:

Multi-channel DataWriters (p. 111) for additional details on reliability with MultChannel DataWriters.

6.172.2.2 System::Int32 DDS::RtpsReliableWriterProtocol_t::high_watermark

When the number of unacknowledged samples in the cache of a reliable writer climbs above this threshold, the DDS::StatusKind::RELIABLE_WRITER_CACHE_CHANGED_STATUS is considered to have changed.

This value is measured in units of samples except with batching configurations in non MultiChannel DataWriters 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. 686),

For **DDS::DiscoveryConfigQosPolicy::publication_writer** (p. 569)

high_watermark<=DDS::DomainParticipantQos::resource_limits.local_writer_allocation.max_count

For **DDS::DiscoveryConfigQosPolicy::subscription_writer** (p. 569)

high_watermark<=DDS::DomainParticipantQos::resource_limits.local_reader_allocation.max_count

In **DDS::DataWriterQos::protocol** (p. 550),

For **DDS::DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 531)

high_watermark<=**DDS::ResourceLimitsQosPolicy::max_samples**

(p. 1112) if batching is disabled or the **DataWriter** (p. 499) is a Multi-Channel **DataWriter** (p. 499). Otherwise,

high_watermark<=**DDS::DataWriterResourceLimitsQosPolicy::max_batches** (p. 555)

[default] 1

[range] [1, 100 million] or DDS::LENGTH_UNLIMITED, > low_watermark <= *maximum* which depends on the container policy

See also:

Multi-channel DataWriters (p. 111) for additional details on reliability with MultChannel DataWriters.

6.172.2.3 Duration_t DDS::RtpsReliableWriterProtocol_t::heartbeat_period

The period at which to send heartbeats.

A reliable writer will send periodic heartbeats at this rate.

[default] 3 seconds

[range] [1 nanosec,1 year], >= **DDS::RtpsReliableWriterProtocol_t::fast_heartbeat_period** (p. 1132), >= **DDS::RtpsReliableWriterProtocol_t::late_joiner_heartbeat_period** (p. 1133)

6.172.2.4 Duration_t DDS::RtpsReliableWriterProtocol_t::fast_heartbeat_period

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 beneath 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] 3 seconds

[range] [1 nanosec,1 year], <= **DDS::RtpsReliableWriterProtocol_t::heartbeat_period** (p. 1132)

6.172.2.5 Duration_t DDS::RtpsReliableWriterProtocol_t::late_joiner_heartbeat_period

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.

[default] 3 seconds

[range] [1 nanosec,1 year], <= **DDS::RtpsReliableWriterProtocol_t::heartbeat_period** (p. 1132)

6.172.2.6 System::Int32 DDS::RtpsReliableWriterProtocol_t::max_heartbeat_retries

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] 10

[range] [1, 1 million] or DDS::LENGTH_UNLIMITED

6.172.2.7 System::Int32 DDS::RtpsReliableWriterProtocol_t::heartbeats_per_max_samples

The number of heartbeats per send queue.

If batching is disabled or the **DataWriter** (p. 499) is a MultiChannel **DataWriter** (p. 499): a piggyback heartbeat will be sent every [**DDS::ResourceLimitsQosPolicy::max_samples** (p. 1112)/heartbeats_per_max_samples] number of samples.

Otherwise: a piggyback heartbeat will be sent every [**DDS::DataWriterResourceLimitsQosPolicy::max_batches** (p. 555)/heartbeats_per_max_samples] number of batches.

If set to zero, no piggyback heartbeat will be sent. If maximum is DDS::LENGTH_UNLIMITED, 100 million is assumed as the maximum value in the calculation.

[default] 8

[range] [0, 100 million]

- For **DDS::DiscoveryConfigQosPolicy::publication_writer** (p. 569):

  heartbeats_per_max_samples<= DDS::DomainParticipantQos::resource_limits.local_writer_allocation.max_count

- For **DDS::DiscoveryConfigQosPolicy::subscription_writer** (p. 569):

  heartbeats_per_max_samples<= DDS::DomainParticipantQos::resource_limits.local_reader_allocation.max_count

- For **DDS::DataWriterProtocolQosPolicy::rtps_reliable_writer** (p. 531):

  heartbeats_per_max_samples<= **DDS::ResourceLimitsQosPolicy::max_samples** (p. 1112) if batching is disabled or the **DataWriter** (p. 499) is a MultiChannel **DataWriter** (p. 499). Otherwise:

  heartbeats_per_max_samples<= **DDS::DataWriterResourceLimitsQosPolicy::max_batches** (p. 555).

6.172.2.8 Duration_t DDS::RtpsReliableWriterProtocol_t::min_nack_response_delay

The minimum delay to respond to a NACK.

When a reliable writer receives a NACK from a remote reader, the writer can choose to delay a while before it sends repair samples or a heartbeat. This sets the value of the minimum delay.

[default] 0 seconds

[range] [0,1 day], <= max_nack_response_delay

6.172.2.9 Duration_t DDS::RtpsReliableWriterProtocol_t::max_nack_response_delay

The maximum delay to respond to a nack.

This set the value of maximum delay between receiving a NACK and sending repair samples or a heartbeat.

[**default**] The default value depends on the container policy:

For **DDS::DiscoveryConfigQosPolicy** (p. 563) : 0 seconds

For **DDS::DataWriterProtocolQosPolicy** (p. 529) : 0.2 seconds

[**range**] [0,1 day], >= min_nack_response_delay

6.172.2.10 Duration_t DDS::RtpsReliableWriterProtocol_t::nack_suppression_duration

The duration for ignoring consecutive NACKs that may trigger redundant repairs.

A reliable writer may receive consecutive NACKs 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 are ignored to prevent redundant repairs from being sent.

[**default**] 0 seconds

[**range**] [0,1 day],

6.172.2.11 System::Int32 DDS::RtpsReliableWriterProtocol_t::max_bytes_per_nack_response

The maximum total message size when resending dropped samples.

As part of the reliable communication protocol, data writers send heartbeat (HB) messages to their data readers. Each HB message contains the sequence number of the most recent sample sent by the data writer.

In response, a data reader sends an acknowledgement (ACK) message, indicating what sequence numbers it did not receive, if any. If the data reader is missing some samples, the data writer will send them again.

max_bytes_per_nack_response determines the maximum size of the message sent by the data writer in response to an ACK. This message may contain multiple samples.

If max_bytes_per_nack_response is larger than the maximum message size supported by the underlying transport, RTI Data Distribution Service will send multiple messages. If the total size of all samples that need to be resent is larger than max_bytes_per_nack_response, the remaining samples will be resent the next time an ACK arrives.

[**default**] 131072

[**range**] [0, 1 GB]

6.172.2.12 Duration_t DDS::RtpsReliableWriterProtocol_t::disable_positive_acks_min_sample_keep_duration

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. 532) = true) or a data reader (**DDS::DataReaderProtocolQosPolicy::disable_positive_acks** (p. 468) = true), 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. 1136)

6.172.2.13 Duration_t DDS::RtpsReliableWriterProtocol_t::disable_positive_acks_max_sample_keep_duration

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. 532) = true) or a data reader (**DDS::DataReaderProtocolQosPolicy::disable_positive_acks** (p. 468) = true), 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. 1136)

6.172.2.14 System::Int32 DDS::RtpsReliableWriterProtocol_t::disable_positive_acks_decrease_sample_keep_duration_factor

Controls rate of contraction of dynamic sample keep duration.

Used when **DDS::RtpsReliableWriterProtocol_t::disable_positive_acks_enable_adaptive_sample_keep_duration** (p. 1140) = true.

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

6.172.2.15 System::Int32 DDS::RtpsReliableWriterProtocol_t::disable_positive_acks_increase_sample_keep_duration_factor

Controls rate of growth of dynamic sample keep duration.

Used when **DDS::RtpsReliableWriterProtocol_t::disable_positive_acks_enable_adaptive_sample_keep_duration** (p. 1140) = true.

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

6.172.2.16 System::Int32 DDS::RtpsReliableWriterProtocol_t::min_send_window_size

Minimum size of send window of unacknowledged samples.

A **DDS::DataWriter** (p. 499) 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 both min_send_window_size and max_send_window_size are DDS::LENGTH_UNLIMITED, then **DDS::ResourceLimitsQosPolicy::max_samples** (p. 1112) serves as the effective send window limit. When **DDS::ResourceLimitsQosPolicy::max_samples** (p. 1112) is less than max_send_window_size, then it serves as the effective max send window; if it is also less than min_send_window_size, then effectively both min and max send window sizes are equal to max samples.

[**default**] 32

[**range**] > 0, <= max_send_window_size, or DDS::LENGTH_UNLIMITED

See also:

DDS::RtpsReliableWriterProtocol_t::max_send_window_size

(p. 1138)

6.172.2.17 System::Int32 DDS::RtpsReliableWriterProtocol_t::max_send_window_size

Maximum size of send window of unacknowledged samples.

A **DDS::DataWriter** (p. 499) 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 both min_send_window_size and max_send_window_size are DDS::LENGTH_UNLIMITED, then **DDS::ResourceLimitsQosPolicy::max_samples** (p. 1112) serves as the effective send window limit. When **DDS::ResourceLimitsQosPolicy::max_samples** (p. 1112) is less than max_send_window_size, then it serves as the effective max send window; if it is also less than min_send_window_size, then effectively both min and max send window sizes are equal to max samples.

[default] 256

[range] > 0, >= min_send_window_size, or DDS::LENGTH_UNLIMITED

See also:

DDS::RtpsReliableWriterProtocol_t::min_send_window_size (p. 1137)

6.172.2.18 Duration_t DDS::RtpsReliableWriterProtocol_t::send_window_update_period

Period in which send window may be dynamically changed.

The **DDS::DataWriter** (p. 499)'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] 3 seconds

[range] > [0,1 year]

See also:

DDS::RtpsReliableWriterProtocol_t::send_window_increase_factor (p. 1139), **DDS::RtpsReliableWriterProtocol_t::send_window_decrease_factor** (p. 1139)

6.172.2.19 System::Int32 DDS::RtpsReliableWriterProtocol_t::send_window_increase_factor

Increases send window size by this percentage when reacting dynamically to network conditions.

The **DDS::DataWriter** (p. 499)'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. 1138), **DDS::RtpsReliableWriterProtocol_t::send_window_decrease_factor** (p. 1139)

6.172.2.20 System::Int32 DDS::RtpsReliableWriterProtocol_t::send_window_decrease_factor

Decreases send window size by this percentage when reacting dynamically to network conditions.

The **DDS::DataWriter** (p. 499)'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] 70

[range] [0, 100]

See also:

DDS::RtpsReliableWriterProtocol_t::send_window_update_period (p. 1138), **DDS::RtpsReliableWriterProtocol_t::send_window_increase_factor** (p. 1139)

6.172.3 Property Documentation

6.172.3.1 System:: Boolean DDS::RtpsReliableWriterProtocol_t::inactivate_nonprogressing_readers [get, set]

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. 1133) have been sent without receiving any ack/nacks back.

By setting this true, 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. 1133) 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] false

6.172.3.2 System:: Boolean DDS::RtpsReliableWriterProtocol_t::disable_positive_acks_enable_adaptive_sample_keep_duration [get, set]

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. 1140) = true, 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. 1137). 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. 1136).

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. 1136) and **DDS::RtpsReliableWriterProtocol_t::disable_positive_acks_max_sample_keep_duration** (p. 1136), respectively.

When **DDS::RtpsReliableWriterProtocol_t::disable_positive_acks_enable_adaptive_sample_keep_duration** (p. 1140) = false, the sample keep duration is set to **DDS::RtpsReliableWriterProtocol_t::disable_positive_acks_min_sample_keep_duration** (p. 1136) .

[**default**] true

6.173 DDS::RtpsWellKnownPorts_t Struct Reference

RTPS well-known port mapping configuration.

`#include <managed_infrastructure.h>`

Public Attributes

- System::Int32 **port_base**
  The base port offset.
- System::Int32 **domain_id_gain**
  Tunable domain gain parameter.
- System::Int32 **participant_id_gain**
  Tunable participant gain parameter.
- System::Int32 **builtin_multicast_port_offset**
  Additional offset for **metatraffic** *multicast port.*
- System::Int32 **builtin_unicast_port_offset**
  Additional offset for **metatraffic** *unicast port.*
- System::Int32 **user_multicast_port_offset**
  Additional offset for **usertraffic** *multicast port.*
- System::Int32 **user_unicast_port_offset**
  Additional offset for **usertraffic** *unicast port.*

Properties

- static **RtpsWellKnownPorts_t RTI_BACKWARDS_COMPATIBLE_RTPS_WELL_KNOWN_PORTS** `[get]`
  Assign to use well-known port mappings which are compatible with previous versions of the RTI Data Distribution Service middleware.
- static **RtpsWellKnownPorts_t INTEROPERABLE_RTPS_WELL_KNOWN_PORTS** `[get]`
  Assign to use well-known port mappings which are compliant with OMG's DDS Interoperability Wire Protocol.

6.173.1 Detailed Description

RTPS well-known port mapping configuration.

RTI Data Distribution Service 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 Data Distribution Service. These parameters are exposed in **DDS::RtpsWellKnownPorts_t** (p. 1142). 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. 1142), the port numbers depend on:

- `domain_id`, as specified in **DDS::DomainParticipantFactory::create_participant** (p. 665)

- `participant_id`, as specified using **DDS::WireProtocolQosPolicy::participant_id** (p. 1427)

The `domain_id` parameter ensures no port conflicts exist between Participants belonging to different domains. This also means that discovery metatraffic in one domain is not visible to Participants in a different domain. The `participant_id` parameter ensures that unique unicast port numbers are assigned to Participants belonging to the same domain on a given host.

The *metatraffic_unicast_port* is used to exchange discovery metatraffic using unicast.

```
metatraffic_unicast_port = port_base + (domain_id_gain * domain_id) + (participant_id_gain * participant_id) + builtin_u
```

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. 573) on a **DDS::DomainParticipant** (p. 577) 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. 484), **DDS::DataReaderQos::unicast** (p. 484), or by the **DDS::DataWriterQos::unicast** (p. 550) 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
```

The *usertraffic_multicast_port* is used to exchange user data using multicast. The corresponding multicast group addresses can be configured using **DDS::TransportMulticastQosPolicy** (p. 1287).

```
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::RtpsWellKnownPorts_t::INTEROPERABLE_RTPS_WELL_KNOWN_PORTS** (p. 329)).

The OMG's DDS Interoperability Wire Protocol compliant port mapping parameters are *not* backwards compatible with previous versions of the RTI Data Distribution Service 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].

QoS:

DDS::WireProtocolQosPolicy (p. 1423)

6.173.2 Member Data Documentation

6.173.2.1 System::Int32 DDS::RtpsWellKnownPorts_t::port_base

The base port offset.

All mapped well-known ports are offset by this value.

[default] 7400

[range] [>= 1], but resulting ports must be within the range imposed by the underlying transport.

6.173.2.2 System::Int32 DDS::RtpsWellKnownPorts_t::domain_id_gain

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. 577) instances within a single domain occupy a consecutive range of `domain_id_gain` ports. Precisely, all ports occupied by the domain fall within:

```
(port_base + (domain_id_gain * domain_id))
```

and:

```
(port_base + (domain_id_gain * (domain_id + 1)) - 1)
```

Under such a case, the highest `domain_id` is limited only by the underlying transport's maximum port. The highest `participant_id`, however, must satisfy:

```
max_participant_id < (domain_id_gain / participant_id_gain)
```

On the contrary, if `domain_id_gain` <= `participant_id_gain`, it results in a port mapping layout where a given domain's **DDS::DomainParticipant** (p. 577) 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 System::Int32 DDS::RtpsWellKnownPorts_t::participant_id_gain

Tunable participant gain parameter.

Multiplier of the participant_id. See **DDS::RtpsWellKnownPorts_t::domain_id_gain** (p. 1145) 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 System::Int32 DDS::RtpsWellKnownPorts_t::builtin_multicast_port_offset

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 System::Int32 DDS::RtpsWellKnownPorts_t::builtin_unicast_port_offset

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 System::Int32 DDS::RtpsWellKnownPorts_t::user_multicast_port_offset

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 System::Int32 DDS::RtpsWellKnownPorts_t::user_unicast_port_offset

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::SampleInfo Class Reference

Information that accompanies each sample that is `read` or `taken`.

`#include <managed_subscription.h>`

Properties

- ^ **SampleStateKind sample_state** `[get, set]`

  The sample state of the sample.

- ^ **ViewStateKind view_state** `[get, set]`

  The view state of the instance.

- ^ **InstanceStateKind instance_state** `[get, set]`

  The instance state of the instance.

- ^ **Time_t source_timestamp** `[get, set]`

  *The timestamp when the sample was written by a **DataWriter** (p. 499).*

- ^ **InstanceHandle_t instance_handle** `[get, set]`

  Identifies locally the corresponding instance.

- ^ **InstanceHandle_t publication_handle** `[get, set]`

  *Identifies locally the **DataWriter** (p. 499) that modified the instance.*

- ^ System::Int32 **disposed_generation_count** `[get, set]`

  The disposed generation count of the instance at the time of sample reception.

- ^ System::Int32 **no_writers_generation_count** `[get, set]`

  The no writers generation count of the instance at the time of sample reception.

- ^ System::Int32 **sample_rank** `[get, set]`

  The sample rank of the sample.

- ^ System::Int32 **generation_rank** `[get, set]`

  The generation rank of the sample.

- ^ System::Int32 **absolute_generation_rank** `[get, set]`

  The absolute generation rank of the sample.

- ^ System::Boolean **valid_data** `[get, set]`

Indicates whether the `DataSample` *contains data or else it is only used to communicate a change in the* `instance_state` *of the instance.*

- **Time_t reception_timestamp** `[get, set]`

  <<**eXtension**>> (p. 174) *The timestamp when the sample was committed by a* ***DataReader*** *(p. 433).*

- **SequenceNumber_t publication_sequence_number** `[get, set]`

  <<**eXtension**>> (p. 174) *The publication sequence number.*

- **SequenceNumber_t reception_sequence_number** `[get, set]`

  <<**eXtension**>> (p. 174) *The reception sequence number when sample was committed by a* ***DataReader*** *(p. 433)*

6.174.1 Detailed Description

Information that accompanies each sample that is `read` or `taken`.

6.174.2 Interpretation of the SampleInfo

The **DDS::SampleInfo** (p. 1148) 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)
- the `valid_data` 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::InstanceStateKind::ALIVE_INSTANCE_STATE** (p. 908)) 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. 433). This rank provides a preview of what is available within the **DDS::DataReader** (p. 433).

- The `source_timestamp` of the sample. This is the timestamp provided by the **DDS::DataWriter** (p. 499) at the time the sample was produced.

6.174.3 Interpretation of the SampleInfo disposed_generation_count and no_writers_generation_count

For each instance, RTI Data Distribution Service internally maintains two counts, the **DDS::SampleInfo::disposed_generation_count** (p. 1153) and **DDS::SampleInfo::no_writers_generation_count** (p. 1154), relative to each **DataReader** (p. 433):

- The **DDS::SampleInfo::disposed_generation_count** (p. 1153) and **DDS::SampleInfo::no_writers_generation_count** (p. 1154) are initialized to zero when the **DDS::DataReader** (p. 433) first detects the presence of a never-seen-before instance.

- The **DDS::SampleInfo::disposed_generation_count** (p. 1153) is incremented each time the instance_state of the corresponding instance changes from **DDS::InstanceStateKind::NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 909) to **DDS::InstanceStateKind::ALIVE_INSTANCE_STATE** (p. 908).

- The **DDS::SampleInfo::no_writers_generation_count** (p. 1154) is incremented each time the instance_state of the corresponding instance changes from **DDS::InstanceStateKind::NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 909) to **DDS::InstanceStateKind::ALIVE_INSTANCE_STATE** (p. 908).

- These 'generation counts' are reset to zero when the instance resource is reclaimed.

The **DDS::SampleInfo::disposed_generation_count** (p. 1153) and **DDS::SampleInfo::no_writers_generation_count** (p. 1154) available in the **DDS::SampleInfo** (p. 1148) capture a snapshot of the corresponding counters at the time the sample was received.

6.174.4 Interpretation of the SampleInfo sample_rank, generation_rank and absolute_generation_rank

The **DDS::SampleInfo::sample_rank** (p. 1154) and **DDS::SampleInfo::generation_rank** (p. 1154) available in the **DDS::SampleInfo** (p. 1148) are computed based solely on the actual samples in the ordered collection returned by read or take.

- The **DDS::SampleInfo::sample_rank** (p. 1154) indicates the number of samples of the same instance that follow the current one in the collection.
- The **DDS::SampleInfo::generation_rank** (p. 1154) available in the **DDS::SampleInfo** (p. 1148) 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. 1154) 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. 1155) available in the **DDS::SampleInfo** (p. 1148) indicates the difference in "generations" between the sample (S) and the Most Recent Sample of the same instance that the middleware has received (MRS). That is, it counts the number of times the instance transitioned from not-alive to alive in the time from the reception of the S to the time when the read or take was called.

```
absolute_generation_rank = (MRS.disposed_generation_count
                                + MRS.no_writers_generation_count)
                            - (S.disposed_generation_count
                                + S.no_writers_generation_count)
```

6.174.5 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. 1148), 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. 1161), **DDS::InstanceStateKind** (p. 907), **DDS::ViewStateKind** (p. 1409), **DDS::SampleInfo::valid_data** (p. 1155)

6.174.6 Property Documentation

6.174.6.1 SampleStateKind DDS::SampleInfo::sample_state [get, set]

The sample state of the sample.

Indicates whether or not the corresponding data sample has already been read.

See also:

DDS::SampleStateKind (p. 1161)

6.174.6.2 ViewStateKind DDS::SampleInfo::view_state [get, set]

The view state of the instance.

Indicates whether the **DDS::DataReader** (p. 433) has already seen samples for the most-current generation of the related instance.

See also:

DDS::ViewStateKind (p. 1409)

6.174.6.3 InstanceStateKind DDS::SampleInfo::instance_state [get, set]

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. 907)

6.174.6.4 Time_t DDS::SampleInfo::source_timestamp [get, set]

The timestamp when the sample was written by a **DataWriter** (p. 499).

6.174.6.5 InstanceHandle_t DDS::SampleInfo::instance_handle [get, set]

Identifies locally the corresponding instance.

6.174.6.6 InstanceHandle_t DDS::SampleInfo::publication_handle [get, set]

Identifies locally the **DataWriter** (p. 499) that modified the instance.

The `publication_handle` is the same **DDS::InstanceHandle_t** (p. 905) that is returned by the operation **DDS::DataReader::get_matched_publications** (p. 442) and can also be used as a parameter to the operation **DDS::DataReader::get_matched_publication_data** (p. 443).

6.174.6.7 System:: Int32 DDS::SampleInfo::disposed_generation_count [get, set]

The disposed generation count of the instance at the time of sample reception.

Indicates the number of times the instance had become alive after it was disposed explicitly by a **DDS::DataWriter** (p. 499), at the time the sample was received.

See also:

Interpretation of the SampleInfo disposed_generation_count and no_writers_generation_count (p. 1150) **Interpretation of the SampleInfo counters and ranks** (p. 1151)

6.174.6.8 System:: Int32 DDS::SampleInfo::no_writers_generation_count [get, set]

The no writers generation count of the instance at the time of sample reception.

Indicates the number of times the instance had become alive after it was disposed because there were no writers, at the time the sample was received.

See also:

Interpretation of the SampleInfo disposed_generation_count and no_writers_generation_count (p. 1150) **Interpretation of the SampleInfo counters and ranks** (p. 1151)

6.174.6.9 System:: Int32 DDS::SampleInfo::sample_rank [get, set]

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. 1151) **Interpretation of the SampleInfo counters and ranks** (p. 1151)

6.174.6.10 System:: Int32 DDS::SampleInfo::generation_rank [get, set]

The 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 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. 1151) **Interpretation of the SampleInfo counters and ranks** (p. 1151)

6.174.6.11 System:: Int32 DDS::SampleInfo::absolute_generation_rank [get, set]

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. 1151) **Interpretation of the SampleInfo counters and ranks** (p. 1151)

6.174.6.12 System:: Boolean DDS::SampleInfo::valid_data [get, set]

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. 1148) and some Data. However there are situations where a DataSample contains only the **DDS::SampleInfo** (p. 1148) and does not have any associated data. This occurs when the RTI Data Distribution Service 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 Data Distribution Service detects that an instance has no writers and changes the corresponding instance_state to **DDS::InstanceStateKind::NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 909).

The application can distinguish whether a particular DataSample has data by examining the value of the valid_data flag. If this flag is set to true, then the DataSample contains valid Data. If the flag is set to false, 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 false, the application should not access the Data associated with the DataSample, that is, the application should access only the **DDS::SampleInfo** (p. 1148).

6.174.6.13 Time_t DDS::SampleInfo::reception_timestamp [get, set]

<<*eXtension*>> *(p. 174)* The timestamp when the sample was committed by a **DataReader** (p. 433).

6.174.6.14 SequenceNumber_t DDS::SampleInfo::publication_sequence_number [get, set]

<<*eXtension*>> *(p. 174)* The publication sequence number.

6.174.6.15 SequenceNumber_t DDS::SampleInfo::reception_sequence_number [get, set]

<<*eXtension*>> *(p. 174)* The reception sequence number when sample was committed by a **DataReader** (p. 433)

6.175 DDS::SampleInfoSeq Class Reference

Declares IDL `sequence` < **DDS::SampleInfo** (p. 1148) > .

`#include <managed_subscription.h>`

Inheritance diagram for DDS::SampleInfoSeq::

6.175.1 Detailed Description

Declares IDL `sequence` < **DDS::SampleInfo** (p. 1148) > .

See also:

DDS::Sequence (p. 1163)

Examples:

HelloWorld_subscriber.cpp.

6.176 DDS::SampleLostStatus Struct Reference

DDS::StatusKind::SAMPLE_LOST_STATUS

`#include <managed_subscription.h>`

Public Attributes

- System::Int32 **total_count**

  *Total cumulative count of all samples lost across all instances of data published under the **DDS::Topic** (p. 1258).*

- System::Int32 **total_count_change**

  The incremental number of samples lost since the last time the listener was called or the status was read.

6.176.1 Detailed Description

DDS::StatusKind::SAMPLE_LOST_STATUS

Examples:

HelloWorld_subscriber.cpp.

6.176.2 Member Data Documentation

6.176.2.1 System::Int32 DDS::SampleLostStatus::total_count

Total cumulative count of all samples lost across all instances of data published under the **DDS::Topic** (p. 1258).

6.176.2.2 System::Int32 DDS::SampleLostStatus::total_count_change

The incremental number of samples lost since the last time the listener was called or the status was read.

6.177 DDS::SampleRejectedStatus Struct Reference

DDS::StatusKind::SAMPLE_REJECTED_STATUS

`#include <managed_subscription.h>`

Public Attributes

- System::Int32 **total_count**

  *Total cumulative count of samples rejected by the **DDS::DataReader** (p. 433).*

- System::Int32 **total_count_change**

  The incremental number of samples rejected since the last time the listener was called or the status was read.

- **SampleRejectedStatusKind last_reason**

  Reason for rejecting the last sample rejected.

- **InstanceHandle_t last_instance_handle**

  Handle to the instance being updated by the last sample that was rejected.

6.177.1 Detailed Description

DDS::StatusKind::SAMPLE_REJECTED_STATUS

Examples:

HelloWorld_subscriber.cpp.

6.177.2 Member Data Documentation

6.177.2.1 System::Int32 DDS::SampleRejectedStatus::total_count

Total cumulative count of samples rejected by the **DDS::DataReader** (p. 433).

6.177.2.2 System::Int32 DDS::SampleRejectedStatus::total_count_change

The incremental number of samples rejected since the last time the listener was called or the status was read.

6.177.2.3 SampleRejectedStatusKind DDS::SampleRejectedStatus::last_reason

Reason for rejecting the last sample rejected.

See also:

DDS::SampleRejectedStatusKind

6.177.2.4 InstanceHandle_t DDS::SampleRejectedStatus::last_instance_handle

Handle to the instance being updated by the last sample that was rejected.

6.178 DDS::SampleStateKind Struct Reference

Indicates whether or not a sample has ever been read.

`#include <managed_subscription.h>`

Properties

- static **SampleStateKind READ_SAMPLE_STATE** `[get]`

  Sample has been read.

- static **SampleStateKind NOT_READ_SAMPLE_STATE** `[get]`

  Sample has not been read.

- static **SampleStateKind ANY_SAMPLE_STATE** `[get]`

  Any sample state ***DDS::SampleStateKind::READ_SAMPLE_STATE*** *(p. 1161) | **DDS::SampleStateKind::NOT_READ_SAMPLE_STATE** (p. 1162).*

6.178.1 Detailed Description

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. 433). The sample state can be either:

- **DDS::SampleStateKind::READ_SAMPLE_STATE** (p. 1161) indicates that the **DDS::DataReader** (p. 433) has already accessed that sample by means of a read or take operation.

- **DDS::SampleStateKind::NOT_READ_SAMPLE_STATE** (p. 1162) indicates that the **DDS::DataReader** (p. 433) 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.

6.178.2 Property Documentation

6.178.2.1 SampleStateKind DDS::SampleStateKind::READ_SAMPLE_STATE `[static, get]`

Sample has been read.

6.178.2.2 SampleStateKind DDS::SampleStateKind::NOT_READ_SAMPLE_STATE [static, get]

Sample has not been read.

6.179 DDS::Sequence< T > Class Template Reference

<<***interface***>> *(p. 175)* <<***generic***>> *(p. 175)* A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as `Foo` (p. 877).

`#include <managed_sequence.h>`

Inheritance diagram for DDS::Sequence< T >::

Public Member Functions

- System::Boolean **ensure_length** (System::Int32 length, System::Int32 max)

  Set the sequence to the desired length, and resize the sequence if necessary.

- virtual T **get_at** (System::Int32 i)

  Get the `i-th` *element for a* `const` *sequence.*

- virtual void **set_at** (System::Int32 i, T val)

  Set the `i-th` *element of the sequence.*

- void **loan** (array< T >^buffer, System::Int32 new_length)

  Loan a contiguous buffer to this sequence.

- virtual void **unloan** ()

  Return the loaned buffer in the sequence and set the maximum to 0.

- void **from_array** (array< T >^arr)

  Copy elements from an array of elements, resizing the sequence if necessary. The original contents of the sequence (if any) are replaced.

- void **to_array** (array< T >^arr)

  Copy elements to an array of elements. The original contents of the array (if any) are replaced.

- System::Boolean **copy_from** (**Sequence**< T >^src_seq)

  Copy elements from another sequence, resizing the sequence if necessary.

- virtual System::Boolean **copy_from_no_alloc** (**Sequence**< T >^src_seq)

Copy elements from another sequence, only if the destination sequence has enough capacity.

Properties

- System::Int32 **length** `[get, set]`

  The logical length of this sequence.

- virtual System::Int32 **maximum** `[get, set]`

  The current maximum number of elements that can be stored in this sequence.

- array< T >^ **buffer** `[get]`

  Return the contiguous buffer of the sequence.

- System::Boolean **has_ownership** `[get]`

  Return the value of the owned flag.

6.179.1 Detailed Description

template<typename T> class DDS::Sequence< T >

<<***interface***>> *(p. 175)* <<***generic***>> *(p. 175)* A type-safe, ordered collection of elements. The type of these elements is referred to in this documentation as `Foo` (p. 877).

For users who define data types in OMG IDL, this type corresponds to the IDL express `sequence<Foo` (p. 877)`>`.

For any user-data type `Foo` (p. 877) that an application defines for the purpose of data-distribution with RTI Data Distribution Service, a `FooSeq` (p. 880) is generated. We refer to an IDL `sequence<Foo` (p. 877)`> as FooSeq` (p. 880).

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 **DDS::Sequence::maximum** (p. 1172) operation.

- The 'length' represents the actual number of elements it currently holds. It is returned by the **DDS::Sequence::length** (p. 1171) operation.

^ The 'owned' flag represents whether the sequence owns the underlying buffer. It is returned by the **DDS::Sequence::has_ownership** (p. 1173) 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 == true, the sequence has ownership on the buffer. It is then responsible for destroying the buffer when the sequence is destroyed.
- If the owned == false, 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 == true

See also:

DDS::TypedDataWriter (p. 1368), **DDS::TypedDataReader** (p. 1338), **FooTypeSupport** (p. 884), **rtiddsgen** (p. 196)

6.179.2 Member Function Documentation

6.179.2.1 template<typename T> System::Boolean DDS::Sequence< T >::ensure_length (System::Int32 *length*, System::Int32 *max*)

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 maximum by freeing and re-allocating the buffer. 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`
owned == true if sequence needs to be resized

Postcondition:

length == `length`
maximum == `max` if resized

Parameters:

length <<*in*>> *(*p. *175)* The new length that should be set. Must be >= 0.

max <<*in*>> *(*p. *175)* If sequence need to be resized, this is the maximum that should be set. `max` >= `length`

Returns:

true on success, false if the preconditions are not met. In that case the sequence is not modified.

6.179.2.2 template<typename T> virtual T DDS::Sequence< T >::get_at (System::Int32 *i*) [virtual]

Get the `i-th` element for a `const` sequence.

Parameters:

i index of element to access, must be >= 0 and less than **DDS::Sequence::length** (p. 1171)

Returns:

the `i-th` element

Reimplemented in **DDS::LoanableSequence< E >** (p. 965), **DDS::LoanableSequence< M^ >** (p. 965), **DDS::LoanableSequence< DDS::SampleInfo^ >** (p. 965), **DDS::LoanableSequence< DDS::TopicBuiltinTopicData^ >** (p. 965), **DDS::LoanableSequence< DDS::PublicationBuiltinTopicData^ >** (p. 965), **DDS::LoanableSequence< Foo^ >** (p. 965), **DDS::LoanableSequence< DDS::ParticipantBuiltinTopicData^ >** (p. 965), and **DDS::LoanableSequence< DDS::SubscriptionBuiltinTopicData^ >** (p. 965).

6.179.2.3 template<typename T> virtual void DDS::Sequence< T >::set_at (System::Int32 *i*, T *val*) [virtual]

Set the `i-th` element of the sequence.

Parameters:

i index of element to access, must be >= 0 and less than **DDS::Sequence::length** (p. 1171)

val <<*in*>> *(*p. *175)* value to be set

6.179.2.4 template<typename T> void DDS::Sequence< T >::loan (array< T >^ *buffer*, System::Int32 *new_length*)

Loan a contiguous buffer to this sequence.

This operation changes the `owned` flag of the sequence to false and also sets the underlying buffer used by the sequence. See the user's manual for more information about sequences and memory ownership.

Use this method if you want to manage the memory used by the sequence yourself. You must provide an array of elements and integers indicating how many elements are allocated in that array (i.e. the maximum) and how many elements are valid (i.e. the length). The sequence will subsequently use the memory you provide and will not permit it to be freed by a call to **DDS::Sequence::maximum** (p. 1172).

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 Data Distribution Service will return a sequence to you that has a loan; those cases are documented as such. For example, if you call **DDS::TypedDataReader::read** (p. 1341) or **DDS::TypedDataReader::take** (p. 1342) and pass in sequences with no loan and no memory allocated, RTI Data Distribution Service will loan memory to your sequences which must be unloaned with **DDS::TypedDataReader::return_loan** (p. 1364). See the documentation of those methods for more information.

Precondition:

DDS::Sequence::maximum (p. 1172) == 0; i.e. the sequence has no memory allocated to it.
DDS::Sequence::has_ownership (p. 1173) == true; i.e. the sequence does not already have an outstanding loan

Postcondition:

The sequence will store its elements in the buffer provided.
DDS::Sequence::has_ownership (p. 1173) == false
DDS::Sequence::length (p. 1171) == new_length
DDS::Sequence::maximum (p. 1172) == new_max

Parameters:

buffer The new buffer that the sequence will use. Must point to enough memory to hold new_max elements of type **Foo** (p. 877). It may be NULL if new_max == 0.

new_length The desired new length for the sequence.

Returns:

true if `buffer` is successfully loaned to this sequence or false otherwise.

Failure only occurs due to failing to meet the pre-conditions. Upon failure the sequence remains unmodified.

See also:

DDS::Sequence::unloan (p. 1168), DDS::Sequence::loan_discontiguous

6.179.2.5 template<typename T> virtual void DDS::Sequence< T >::unloan () [virtual]

Return the loaned buffer in the sequence and set the maximum to 0.

This method affects only the state of this sequence; it does not change the contents of the buffer in any way.

Only the user who originally loaned a buffer should return that loan, as the user may have dependencies on that memory known only to them. Unloaning someone else's buffer may cause unspecified problems. For example, suppose a sequence is loaning memory from a custom memory pool. A user of the sequence likely has no way to release the memory back into the pool, so unloaning the sequence buffer would result in a resource leak. If the user were to then re-loan a different buffer, the original creator of the sequence would have no way to discover, when freeing the sequence, that the loan no longer referred to its own memory and would thus not free the user's memory properly, exacerbating the situation and leading to undefined behavior.

Precondition:

owned == false

Postcondition:

owned == true
maximum == 0

Returns:

true if the preconditions were met. Otherwise false. The function only fails if the pre-conditions are not met, in which case it leaves the sequence unmodified.

See also:

DDS::Sequence<T>::loan (p. 1167)(array<T>^, System::Int32), DDS::Sequence::loan_discontiguous, **DDS::Sequence::maximum** (p. 1172)

Reimplemented in **DDS::LoanableSequence< E >** (p. 966), **DDS::LoanableSequence< M^ >** (p. 966), **DDS::LoanableSequence<**

DDS::SampleInfo^ > (p. 966), **DDS::LoanableSequence< DDS::TopicBuiltinTopicData^ >** (p. 966), **DDS::LoanableSequence< DDS::PublicationBuiltinTopicData^ >** (p. 966), **DDS::LoanableSequence< Foo^ >** (p. 966), **DDS::LoanableSequence< DDS::ParticipantBuiltinTopicData^ >** (p. 966), and **DDS::LoanableSequence< DDS::SubscriptionBuiltinTopicData^ >** (p. 966).

6.179.2.6 template<typename T> void DDS::Sequence< T >::from_array (array< T >^ *arr*)

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 == true

Postcondition:

this::length == `length`
this[i] == array[i] for 0 <= i < `length`
this::owned == true

Parameters:

arr <<*in*>> *(p. 175)* The array of elements to be copy elements from

Returns:

true if the array was successfully copied; false otherwise.

Note:

If the pre-conditions are not met, the method will print a message to stdout and leave this sequence unchanged.

6.179.2.7 template<typename T> void DDS::Sequence< T >::to_array (array< T >^ *arr*)

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:

arr *<<in>>* *(p. 175)* The array of elements to be filled with elements from this sequence

Returns:

true if the elements of the sequence were successfully copied; false otherwise.

6.179.2.8 template<typename T> System::Boolean DDS::Sequence< T >::copy_from (Sequence< T >^ *src_seq*)

Copy elements from another sequence, resizing the sequence if necessary.

This method invokes **DDS::Sequence::copy_from** (p. 1170)(DDS::Sequence<T>^) after ensuring that the sequence has enough capacity to hold the elements to be copied.

Parameters:

src_seq *<<in>>* *(p. 175)* the sequence from which to copy

See also:

DDS::Sequence::copy_from (p. 1170)(DDS::Sequence<T>^)

6.179.2.9 template<typename T> virtual System::Boolean DDS::Sequence< T >::copy_from_no_alloc (Sequence< T >^ *src_seq*) `[virtual]`

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 == true

Postcondition:

this::length == src_seq::length
this[i] == src_seq[i] for 0 <= i < target_seq::length
this::owned == true

Parameters:

src_seq *<<in>>* *(p. 175)* the sequence from which to copy

Returns:

true if the sequence was successfully copied; false otherwise.

Note:

If the pre-conditions are not met, the operator will print a message to stdout and leave this sequence unchanged.

See also:

DDS::Sequence::copy_from_no_alloc (p. 1170)(DDS::Sequence<T>^)

Reimplemented in **DDS::LoanableSequence< E >** (p. 965), **DDS::LoanableSequence< M^ >** (p. 965), **DDS::LoanableSequence< DDS::SampleInfo^ >** (p. 965), **DDS::LoanableSequence< DDS::TopicBuiltinTopicData^ >** (p. 965), **DDS::LoanableSequence< DDS::PublicationBuiltinTopicData^ >** (p. 965), **DDS::LoanableSequence< Foo^ >** (p. 965), **DDS::LoanableSequence< DDS::ParticipantBuiltinTopicData^ >** (p. 965), and **DDS::LoanableSequence< DDS::SubscriptionBuiltinTopicData^ >** (p. 965).

6.179.3 Property Documentation

6.179.3.1 template<typename T> System:: Int32 DDS::Sequence< T >::length [get, set]

The logical length of this sequence.

Getting the property:

Get the length that was last set, or zero if the length has never been set.

Setting the property:

Change the length of this sequence. This method does not allocate/deallocate memory.

The new length must not exceed the maximum of this sequence as returned by the **DDS::Sequence::maximum** (p. 1172) operation. (Note that, if necessary, the maximum of this sequence can be increased manually by using the **DDS::Sequence::maximum** (p. 1172) operation.)

The elements of the sequence are not modified by this operation. If the new length is larger than the original length, the new elements will be uninitialized; if the length is decreased, the old elements that are beyond the new length will physically remain in the sequence but will not be accessible.

Postcondition:

length = new_length.

Parameters:

new_length the new desired length. This value must be non-negative and cannot exceed maximum of the sequence. In other words 0 <= new_length <= maximum

6.179.3.2 template<typename T> virtual System:: Int32 DDS::Sequence< T >::maximum [get, set]

The current maximum number of elements that can be stored in this sequence.

Getting the property:

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 **DDS::Sequence::maximum** (p. 1172) operation.

See also:

DDS::Sequence::length (p. 1171)

Setting the property:

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 == true

Postcondition:

owned == true
length == MINIMUM(original length, new_max)

Parameters:

new_max Must be >= 0.

Reimplemented in **DDS::LoanableSequence< E >** (p. 967), **DDS::LoanableSequence< M^ >** (p. 967), **DDS::LoanableSequence< DDS::SampleInfo^ >** (p. 967), **DDS::LoanableSequence< DDS::TopicBuiltinTopicData^ >** (p. 967), **DDS::LoanableSequence< DDS::PublicationBuiltinTopicData^ >** (p. 967), **DDS::LoanableSequence< Foo^ >** (p. 967), **DDS::LoanableSequence< DDS::ParticipantBuiltinTopicData^ >** (p. 967), and **DDS::LoanableSequence< DDS::SubscriptionBuiltinTopicData^ >** (p. 967).

6.179.3.3 template<typename T> array< T>^ DDS::Sequence< T >::buffer [get]

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 **DDS::Sequence::length** (p. 1171) of the sequence are valid.

This property is real-only.

This method provides almost no encapsulation of the sequence's underlying implementation. Certain operations, such as **DDS::Sequence::maximum** (p. 1172), may render the buffer invalid. In light of these caveats, this operation should be used with care.

Returns:

buffer that stores contiguous elements in sequence.

6.179.3.4 template<typename T> System:: Boolean DDS::Sequence< T >::has_ownership [get]

Return the value of the owned flag.

This property is real-only.

Returns:

true if sequence owns the underlying buffer, or false if it has an outstanding loan.

6.180 DDS::SequenceNumber_t Struct Reference

Type for *sequence* number representation.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- ^ static Int32 **sequence_number_compare** (**SequenceNumber_t**^ sn1, **SequenceNumber_t**^ sn2)

  Compares two sequence numbers.

Public Attributes

- ^ Int32 **high**

  The most significant part of the sequence number.

- ^ UInt32 **low**

  The least significant part of the sequence number.

Properties

- ^ static **SequenceNumber_t SEQUENCE_NUMBER_UNKNOWN** `[get]`

  Unknown sequence number.

- ^ static **SequenceNumber_t SEQUENCE_NUMBER_ZERO** `[get]`

  Zero value for the sequence number.

- ^ static **SequenceNumber_t SEQUENCE_NUMBER_MAX** `[get]`

  Highest, most positive value for the sequence number.

6.180.1 Detailed Description

Type for *sequence* number representation.

Represents a 64-bit sequence number.

6.180.2 Member Data Documentation

6.180.2.1 Int32 DDS::SequenceNumber_t::high

The most significant part of the sequence number.

6.180.2.2 UInt32 DDS::SequenceNumber_t::low

The least significant part of the sequence number.

6.181 DDS::ShmemTransport Interface Reference

Built-in transport plug-in for inter-process communications using shared memory.

`#include <managed_transport.h>`

6.181.1 Detailed Description

Built-in transport plug-in for inter-process communications using shared memory.

This plugin uses System Shared Memory to send messages between processes on the same node.

The transport plugin has exactly one "receive interface"; since the `address_bit_count` is 0, it can be assigned any address. Thus the interface is located by the "network address" associated with the transport plugin.

6.181.2 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.

6.181.3 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 Data Distribution Service 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.

6.181.4 Shared Resource Keys

The transport uses the **shared memory segment keys**, given by the formula below.

```
0x400000 + port
```

The transport also uses signaling **shared semaphore keys** given by the formula below.

```
0x800000 + port
```

The transport also uses mutex **shared semaphore keys** given by the formula below.

```
0xb00000 + port
```

wher the `port` is a function of the `domain_id` and the `participant_id`, as described in **DDS::WireProtocolQosPolicy::participant_id** (p. 1427)

See also:

DDS::WireProtocolQosPolicy::participant_id (p. 1427)
DDS::Transport_Support::set_builtin_transport_property()

6.181.5 Creating and Registering Shared Memory Transport Plugin

RTI Data Distribution Service can implicitly create this plugin and register with the **DDS::DomainParticipant** (p. 577) if this transport is specified in **DDS::TransportBuiltinQosPolicy** (p. 1285).

To specify the properties of the builtin shared memory transport that is implicitly registered, you can either:

- call DDS::Transport_Support::set_builtin_transport_property or
- specify the pre-defined property names in **DDS::PropertyQosPolicy** (p. 1023) associated with the **DDS::DomainParticipant** (p. 577). (see **Shared Memory Transport Property Names in Property QoS Policy of Domain Participant** (p. 1179)).

Builtin transport plugin properties specified in **DDS::PropertyQosPolicy** (p. 1023) always overwrite the ones specified through DDS::Transport_Support::set_builtin_transport_property(). 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 Data Distribution Service. See **Built-in Transport Plugins** (p. 122) for details on when a builtin transport is registered.

6.181.6 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::PropertyQosPolicy** (p. 1023) of a **DDS::DomainParticipant** (p. 577) to configure the builtin shared memory transport plugin.

Name	Descriptions
dds.transport.shmem.builtin.parent.address_bit_count	See DDS::Transport_Property_t::address_bit_count
dds.transport.shmem.builtin.parent.properties_bitmap	See DDS::Transport_Property_t::properties_bitmap
dds.transport.shmem.builtin.parent.gather_send_buffer_count_max	See DDS::Transport_Property_t::gather_send_buffer_count_max
dds.transport.shmem.builtin.parent.message_size_max	See DDS::Transport_Property_t::message_size_max
dds.transport.shmem.builtin.parent.allow_interfaces	See DDS::Transport_Property_t::allow_interfaces_list and DDS::Transport_Property_t::allow_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.shmem.builtin.parent.deny_interfaces	See DDS::Transport_Property_t::deny_interfaces_list and DDS::Transport_Property_t::deny_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.shmem.builtin.parent.allow_multicast_interfaces	See DDS::Transport_Property_t::allow_multicast_interfaces_list and DDS::Transport_Property_t::allow_multicast_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.shmem.builtin.parent.deny_multicast_interfaces	See DDS::Transport_Property_t::deny_multicast_interfaces_list and DDS::Transport_Property_t::deny_multicast_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.shmem.builtin.received_message_count_max	See DDS::ShmemTransport_Property_t::received_message_count_max
dds.transport.shmem.builtin.receive_buffer_size	See DDS::ShmemTransport_Property_t::receive_buffer_size

Table 6.6: *Property Strings for Shared Memory Transport*

6.182 DDS::ShortSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < System::Int16 >.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::ShortSeq::

Public Member Functions

- ^ **ShortSeq** ()

  Constructs an empty sequence of short integers with an initial maximum of zero.

- ^ **ShortSeq** (System::Int32 max)

  Constructs an empty sequence of short integers with the given initial maximum.

- ^ **ShortSeq** (**ShortSeq**^ shorts)

  Constructs a new sequence containing the given shorts.

6.182.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < System::Int16 >.

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

System::Int16
DDS::Sequence (p. 1163)

6.182.2 Constructor & Destructor Documentation

6.182.2.1 DDS::ShortSeq::ShortSeq () `[inline]`

Constructs an empty sequence of short integers with an initial maximum of zero.

6.182.2.2 DDS::ShortSeq::ShortSeq (System::Int32 *max*) [inline]

Constructs an empty sequence of short integers with the given initial maximum.

6.182.2.3 DDS::ShortSeq::ShortSeq (ShortSeq^ *shorts*) [inline]

Constructs a new sequence containing the given shorts.

Parameters:

shorts the initial contents of this sequence

6.183 DDS::StatusCondition Class Reference

<<*interface*>> *(p. 175)* A specific **DDS::Condition** (p. 408) that is associated with each **DDS::Entity** (p. 845).

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::StatusCondition::

Public Member Functions

- ^ **StatusMask get_enabled_statuses** ()

  *Get the list of statuses enabled on an **DDS::Entity** (p. 845).*

- ^ void **set_enabled_statuses** (**StatusMask** mask)

  This operation defines the list of communication statuses that determine the `trigger_value` *of the **DDS::StatusCondition** (p. 1183).*

- ^ **Entity^ get_entity** ()

  *Get the **DDS::Entity** (p. 845) associated with the **DDS::StatusCondition** (p. 1183).*

- ^ virtual System::Boolean **get_trigger_value** () override

  Retrieve the `trigger_value`*.*

6.183.1 Detailed Description

<<*interface*>> *(p. 175)* A specific **DDS::Condition** (p. 408) that is associated with each **DDS::Entity** (p. 845).

The `trigger_value` of the **DDS::StatusCondition** (p. 1183) 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. 1183).

See also:

Status Kinds (p. 238)
DDS::WaitSet (p. 1411), **DDS::Condition** (p. 408)
DDS::Listener (p. 952)

6.183.2 Member Function Documentation

6.183.2.1 StatusMask DDS::StatusCondition::get_enabled_statuses ()

Get the list of statuses enabled on an **DDS::Entity** (p. 845).

Returns:

list of enabled statuses.

6.183.2.2 void DDS::StatusCondition::set_enabled_statuses (StatusMask *mask*)

This operation defines the list of communication statuses that determine the `trigger_value` of the **DDS::StatusCondition** (p. 1183).

This operation may change the `trigger_value` of the **DDS::StatusCondition** (p. 1183).

DDS::WaitSet (p. 1411) objects' behavior depends on the changes of the `trigger_value` of their attached conditions. Therefore, any **DDS::WaitSet** (p. 1411) to which the **DDS::StatusCondition** (p. 1183) is attached is potentially affected by this operation.

If this function is not invoked, the default list of enabled statuses includes all the statuses.

Parameters:

mask *<<in>> (p. 175)* the list of enables statuses (see **Status Kinds** (p. 238))

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.183.2.3 Entity ^ DDS::StatusCondition::get_entity ()

Get the **DDS::Entity** (p. 845) associated with the **DDS::StatusCondition** (p. 1183).

There is exactly one **DDS::Entity** (p. 845) assocated with each **DDS::StatusCondition** (p. 1183).

Returns:

DDS::Entity (p. 845) associated with the **DDS::StatusCondition** (p. 1183).

6.183.2.4 virtual System::Boolean DDS::StatusCondition::get_trigger_value () [override, virtual]

Retrieve the `trigger_value`.

Returns:

the trigger value.

Implements **DDS::Condition** (p. 408).

6.184 DDS::StringDataReader Class Reference

<<***interface***>> *(p. 175)* Instantiates `DataReader` (p. 433) < System::String >.

`#include <managed_stringwrapperSupport.h>`

Inheritance diagram for DDS::StringDataReader::

Public Member Functions

- ^ void **read** (**DDS::StringSeq**^ received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, System::UInt32 sample_states, System::UInt32 view_states, System::UInt32 instance_states)

  Access a collection of data samples from the ***DDS::DataReader*** *(p. 433).*

- ^ void **take** (**DDS::StringSeq**^ received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, System::UInt32 sample_states, System::UInt32 view_states, System::UInt32 instance_states)

  Access a collection of data-samples from the ***DDS::DataReader*** *(p. 433).*

- ^ void **read_w_condition** (**DDS::StringSeq**^ received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, **DDS::ReadCondition**^ condition)

  Accesses via ***DDS::StringDataReader::read*** *(p. 1187) the samples that match the criteria specified in the* ***DDS::ReadCondition*** *(p. 1084).*

- ^ void **take_w_condition** (**DDS::StringSeq**^ received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, **DDS::ReadCondition**^ condition)

  Analogous to ***DDS::StringDataReader::read_w_condition*** *(p. 1188) except it accesses samples via the* ***DDS::StringDataReader::take*** *(p. 1187) operation.*

- ^ System::String^ **read_next_sample** (**DDS::SampleInfo**^ sample_info)

  Copies the next not-previously-accessed data value from the ***DDS::DataReader*** *(p. 433).*

- ^ System::String^ **take_next_sample** (**DDS::SampleInfo**^ sample_info)

  Copies the next not-previously-accessed data value from the ***DDS::DataReader*** *(p. 433).*

^ void **return_loan** (**DDS::StringSeq**^ received_data, **DDS::SampleInfoSeq**^ info_seq)

*Indicates to the **DDS::DataReader** (p. 433) 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. 433).*

6.184.1 Detailed Description

<<***interface***>> *(p. 175)* Instantiates `DataReader (p. 433)` < System::String >.

See also:

DDS::TypedDataReader (p. 1338)
DDS::DataReader (p. 433)

6.184.2 Member Function Documentation

6.184.2.1 void DDS::StringDataReader::read (DDS::StringSeq^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, System::UInt32 *sample_states*, System::UInt32 *view_states*, System::UInt32 *instance_states*) [inline]

Access a collection of data samples from the **DDS::DataReader** (p. 433).

See also:

DDS::TypedDataReader::read (p. 1341)

6.184.2.2 void DDS::StringDataReader::take (DDS::StringSeq^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, System::UInt32 *sample_states*, System::UInt32 *view_states*, System::UInt32 *instance_states*) [inline]

Access a collection of data-samples from the **DDS::DataReader** (p. 433).

See also:

DDS::TypedDataReader::take (p. 1342)

6.184.2.3 void DDS::StringDataReader::read_w_condition (DDS::StringSeq^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, DDS::ReadCondition^ *condition*) [inline]

Accesses via **DDS::StringDataReader::read** (p. 1187) the samples that match the criteria specified in the **DDS::ReadCondition** (p. 1084).

See also:

DDS::TypedDataReader::read_w_condition (p. 1349)

6.184.2.4 void DDS::StringDataReader::take_w_condition (DDS::StringSeq^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, DDS::ReadCondition^ *condition*) [inline]

Analogous to **DDS::StringDataReader::read_w_condition** (p. 1188) except it accesses samples via the **DDS::StringDataReader::take** (p. 1187) operation.

See also:

DDS::TypedDataReader::take_w_condition (p. 1350)

6.184.2.5 System::String ^ DDS::StringDataReader::read_next_sample (DDS::SampleInfo^ *sample_info*) [inline]

Copies the next not-previously-accessed data value from the **DDS::DataReader** (p. 433).

See also:

DDS::TypedDataReader::read_next_sample (p. 1351)

6.184.2.6 System::String ^ DDS::StringDataReader::take_next_sample (DDS::SampleInfo^ *sample_info*) [inline]

Copies the next not-previously-accessed data value from the **DDS::DataReader** (p. 433).

See also:

DDS::TypedDataReader::take_next_sample (p. 1352)

6.184.2.7 void DDS::StringDataReader::return_loan (DDS::StringSeq^ *received_data*, DDS::SampleInfoSeq^ *info_seq*) [inline]

Indicates to the **DDS::DataReader** (p. 433) 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. 433).

See also:

DDS::TypedDataReader::return_loan (p. 1364)

6.185 DDS::StringDataWriter Class Reference

<<***interface***>> *(p. 175)* Instantiates `DataWriter` (p. 499) < System::String >.

`#include <managed_stringwrapperSupport.h>`

Inheritance diagram for DDS::StringDataWriter::

Public Member Functions

- ^ void **write** (System::String^ instance_data, **DDS::InstanceHandle_t**% handle)

  Modifies the value of a string data instance.

- ^ void **write_w_timestamp** (System::String^ instance_data, **DDS::InstanceHandle_t**% handle, **DDS::Time_t**% source_timestamp)

  Performs the same function as ***DDS::StringDataWriter::write*** *(p. 1190) except that it also provides the value for the* `source_timestamp`.

6.185.1 Detailed Description

<<***interface***>> *(p. 175)* Instantiates `DataWriter` (p. 499) < System::String >.

See also:

DDS::TypedDataWriter (p. 1368)
DDS::DataWriter (p. 499)

6.185.2 Member Function Documentation

6.185.2.1 void DDS::StringDataWriter::write (System::String^ *instance_data*, DDS::InstanceHandle_t% *handle*) [inline]

Modifies the value of a string data instance.

See also:

DDS::TypedDataWriter::write (p. 1376)

6.185.2.2 void DDS::StringDataWriter::write_w_timestamp (System::String^ *instance_data*, DDS::InstanceHandle_t% *handle*, DDS::Time_t% *source_timestamp*) [inline]

Performs the same function as **DDS::StringDataWriter::write** (p. 1190) except that it also provides the value for the `source_timestamp`.

See also:

DDS::TypedDataWriter::write_w_timestamp (p. 1378)

6.186 DDS::StringSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < System::String > with value type semantics.

`#include <managed_infrastructure.h>`

Inherits DDS::StringWrapperSeq.

Public Member Functions

- ^ **StringSeq** ()

  Constructs an empty sequence of strings with an initial maximum of zero.

- ^ **StringSeq** (System::Int32 max)

  Constructs an empty sequence of strings with the given initial maximum.

- ^ **StringSeq** (**StringSeq**^ strings)

  Constructs a new sequence containing the given strings.

6.186.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < System::String > with value type semantics.

StringSeq (p. 1192) is a sequence that contains strings.

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

DDS::Sequence (p. 1163)

6.186.2 Constructor & Destructor Documentation

6.186.2.1 DDS::StringSeq::StringSeq () `[inline]`

Constructs an empty sequence of strings with an initial maximum of zero.

6.186.2.2 DDS::StringSeq::StringSeq (System::Int32 *max*) [inline]

Constructs an empty sequence of strings with the given initial maximum.

6.186.2.3 DDS::StringSeq::StringSeq (StringSeq^ *strings*) [inline]

Constructs a new sequence containing the given strings.

Parameters:

strings the initial contents of this sequence

6.187 DDS::StringTypeSupport Class Reference

<<***interface***>> *(p. 175)* String type support.

`#include <managed_stringwrapperSupport.h>`

Inherits DDS::TypedTypeSupport< T >.

Static Public Member Functions

- static System::String^ **get_type_name** ()

  Get the default name for the System::String type.

- static void **print_data** (System::String^ a_data)

  <<**eXtension**>> (p. 174) *Print value of data type to standard out.*

- static void **register_type** (**DDS::DomainParticipant**^ participant, System::String^ type_name)

  Allows an application to communicate to RTI Data Distribution Service the existence of the System::String data type.

- static void **unregister_type** (**DDS::DomainParticipant**^ participant, System::String^ type_name)

  Allows an application to unregister the System::String data type from RTI Data Distribution Service. After calling unregister_type, no further communication using this type is possible.

6.187.1 Detailed Description

<<***interface***>> *(p. 175)* String type support.

6.187.2 Member Function Documentation

6.187.2.1 static System::String ^ DDS::StringTypeSupport::get_type_name () [static]

Get the default name for the System::String type.

Can be used for calling **DDS::StringTypeSupport::register_type** (p. 1195) or creating **DDS::Topic** (p. 1258).

Returns:

default name for the System::String type.

See also:

DDS::StringTypeSupport::register_type (p. 1195)
DDS::DomainParticipant::create_topic (p. 621)

6.187.2.2 static void DDS::StringTypeSupport::print_data (System::String^ *a_data*) [inline, static]

<<eXtension>> (p. 174) Print value of data type to standard out.

The *generated* implementation of the operation knows how to print value of a data type.

Parameters:

a_data *<<in>> (p. 175)* String to be printed.

6.187.2.3 static void DDS::StringTypeSupport::register_type (DDS::DomainParticipant^ *participant*, System::String^ *type_name*) [static]

Allows an application to communicate to RTI Data Distribution Service the existence of the System::String data type.

By default, The System::String built-in type is automatically registered when a **DomainParticipant** (p. 577) is created using the type_name returned by **DDS::StringTypeSupport::get_type_name** (p. 1194). Therefore, the usage of this function is optional and it is only required when the automatic built-in type registration is disabled using the participant property "dds.builtin_type.auto_register".

This method can also be used to register the same **DDS::StringTypeSupport** (p. 1194) with a **DDS::DomainParticipant** (p. 577) using different values for the type_name.

If `register_type` is called multiple times with the same **DDS::DomainParticipant** (p. 577) and `type_name`, the second (and subsequent) registrations are ignored by the operation.

Parameters:

participant *<<in>> (p. 175)* the **DDS::DomainParticipant** (p. 577) to register the data type System::String with. Cannot be null.

type_name *<<in>>* *(p. 175)* the type name under with the data type System::String is registered with the participant; this type name is used when creating a new **DDS::Topic** (p. 1258). (See **DDS::DomainParticipant::create_topic** (p. 621).) The name may not be null or longer than 255 characters.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_PreconditionNotMet** (p. 1123) or **DDS::Retcode_OutOfResources** (p. 1122).

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. 621)

6.187.2.4 static void DDS::StringTypeSupport::unregister_type (DDS::DomainParticipant^ *participant*, System::String^ *type_name*) [static]

Allows an application to unregister the System::String data type from RTI Data Distribution Service. After calling unregister_type, no further communication using this type is possible.

Precondition:

The System::String type with `type_name` is registered with the participant and all **DDS::Topic** (p. 1258) objects referencing the type have been destroyed. If the type is not registered with the participant, or if any **DDS::Topic** (p. 1258) is associated with the type, the operation will fail with **DDS::Retcode_Error** (p. 1116).

Postcondition:

All information about the type is removed from RTI Data Distribution Service. No further communication using this type is possible.

Parameters:

participant *<<in>>* *(p. 175)* the **DDS::DomainParticipant** (p. 577) to unregister the data type System::String from. Cannot be null.

type_name *<<in>>* *(p. 175)* the type name under with the data type System::String is registered with the participant. The name should match a name that has been previously used to register a type with the participant. Cannot be null.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_BadParameter** (p. 1115) or **DDS::Retcode_Error** (p. 1116)

MT Safety:

SAFE.

See also:

DDS::StringTypeSupport::register_type (p. 1195)

6.188 DDS::StructMember Class Reference

A description of a member of a struct.

`#include <managed_typecode.h>`

Public Attributes

- ^ System::String^ **name**

  The name of the struct member.

- ^ **TypeCode^ type**

  The type of the struct member.

- ^ System::Boolean **is_pointer**

  Indicates whether the struct member is a pointer or not.

- ^ System::Int16 **bits**

  Number of bits of a bitfield member.

- ^ System::Boolean **is_key**

  Indicates if the struct member is a key member or not.

6.188.1 Detailed Description

A description of a member of a struct.

See also:

DDS::StructMemberSeq (p. 1200)
DDS::TypeCodeFactory::create_struct_tc (p. 1331)

6.188.2 Member Data Documentation

6.188.2.1 System::String ^ DDS::StructMember::name

The name of the struct member.

Cannot be null.

6.188.2.2 TypeCode ^ DDS::StructMember::type

The type of the struct member.

Cannot be null.

6.188.2.3 System::Boolean DDS::StructMember::is_pointer

Indicates whether the struct member is a pointer or not.

6.188.2.4 System::Int16 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. 65).

6.188.2.5 System::Boolean DDS::StructMember::is_key

Indicates if the struct member is a key member or not.

6.189 DDS::StructMemberSeq Class Reference

Defines a sequence of struct members.

`#include <managed_typecode.h>`

Inheritance diagram for DDS::StructMemberSeq::

6.189.1 Detailed Description

Defines a sequence of struct members.

See also:

DDS::StructMember (p. 1198)
DDS::Sequence (p. 1163)
DDS::TypeCodeFactory::create_struct_tc (p. 1331)

6.190 DDS::Subscriber Class Reference

<<***interface***>> *(p. 175)* A subscriber is the object responsible for actually receiving data from a subscription.

`#include <managed_subscription.h>`

Inheritance diagram for DDS::Subscriber::

Public Member Functions

- ^ void **get_default_datareader_qos** (**DataReaderQos**^ qos)

  *Copies the default **DDS::DataReaderQos** (p. 480) values into the provided **DDS::DataReaderQos** (p. 480) instance.*

- ^ void **set_default_datareader_qos** (**DataReaderQos**^ qos)

  *Sets the default **DDS::DataReaderQos** (p. 480) values for this subscriber.*

- ^ void **set_default_datareader_qos_with_profile** (System::String^ library_name, System::String^ profile_name)

  <<**eXtension**>> (p. 174) *Set the default **DDS::DataReaderQos** (p. 480) values for this subscriber based on the input XML QoS profile.*

- ^ void **set_default_library** (System::String^ library_name)

  <<**eXtension**>> (p. 174) *Sets the default XML library for a **DDS::Subscriber** (p. 1201).*

- ^ System::String^ **get_default_library** ()

  <<**eXtension**>> (p. 174) *Gets the default XML library associated with a **DDS::Subscriber** (p. 1201).*

- ^ void **set_default_profile** (System::String^ library_name, System::String^ profile_name)

  <<**eXtension**>> (p. 174) *Sets the default XML profile for a **DDS::Subscriber** (p. 1201).*

- ^ System::String^ **get_default_profile** ()

  <<**eXtension**>> (p. 174) *Gets the default XML profile associated with a **DDS::Subscriber** (p. 1201).*

- ^ System::String^ **get_default_profile_library** ()

  <<**eXtension**>> (p. 174) *Gets the library where the default XML QoS profile is contained for a **DDS::Subscriber** (p. 1201).*

^ **DataReader**^ **create_datareader** (**ITopicDescription**^ topic, **DataReaderQos**^ qos, **DataReaderListener**^ listener, **StatusMask** mask)

*Creates a **DDS::DataReader** (p. 433) that will be attached and belong to the **DDS::Subscriber** (p. 1201).*

^ **DataReader**^ **create_datareader_with_profile** (**ITopicDescription**^ topic, System::String^ library_name, System::String^ profile_name, **DataReaderListener**^ listener, **StatusMask** mask)

<<**eXtension**>> (p. 174) *Creates a **DDS::DataReader** (p. 433) object using the **DDS::DataReaderQos** (p. 480) associated with the input XML QoS profile.*

^ void **delete_datareader** (**DataReader**^ %a_datareader)

*Deletes a **DDS::DataReader** (p. 433) that belongs to the **DDS::Subscriber** (p. 1201).*

^ void **delete_contained_entities** ()

*Deletes all the entities that were created by means of the "create" operation on the **DDS::Subscriber** (p. 1201).*

^ **DataReader**^ **lookup_datareader** (System::String^ topic_name)

*Retrieves an existing **DDS::DataReader** (p. 433).*

^ void **begin_access** ()

*[**Not supported (optional)**] Indicates that the application is about to access the data samples in any of the **DDS::DataReader** (p. 433) objects attached to the **DDS::Subscriber** (p. 1201).*

^ void **end_access** ()

*[**Not supported (optional)**] Indicates that the application has finished accessing the data samples in **DDS::DataReader** (p. 433) objects managed by the **DDS::Subscriber** (p. 1201).*

^ void **get_datareaders** (**DataReaderSeq**^ readers, System::UInt32 sample_states, System::UInt32 view_states, System::UInt32 instance_states)

*Allows the application to access the **DDS::DataReader** (p. 433) objects that contain samples with the specified* `sample_states`, `view_states` *and* `instance_states`.

^ void **notify_datareaders** ()

*Invokes the operation **DDS::DataReaderListener::on_data_available()** (p. 463) on the **DDS::DataReaderListener** (p. 461) objects attached to contained **DDS::DataReader** (p. 433) entities with DDS::StatusKind::DATA_AVAILABLE_STATUS that is considered changed as described in **Changes in read communication status** (p. 241).*

^ **DomainParticipant**^ **get_participant** ()

*Returns the **DDS::DomainParticipant** (p. 577) to which the **DDS::Subscriber** (p. 1201) belongs.*

^ void **copy_from_topic_qos** (**DataReaderQos**^ datareader_qos, **TopicQos**^ topic_qos)

*Copies the policies in the **DDS::TopicQos** (p. 1280) to the corresponding policies in the **DDS::DataReaderQos** (p. 480).*

^ void **set_qos** (**SubscriberQos**^ qos)

Sets the subscriber QoS.

^ void **set_qos_with_profile** (System::String^ library_name, System::String^ profile_name)

<<**eXtension**>> (p. 174) *Change the QoS of this subscriber using the input XML QoS profile.*

^ void **get_qos** (**SubscriberQos**^ qos)

Gets the subscriber QoS.

^ void **set_listener** (**SubscriberListener**^ l, **StatusMask** mask)

Sets the subscriber listener.

^ **SubscriberListener**^ **get_listener** ()

Get the subscriber listener.

^ virtual void **enable** () override

*Enables the **DDS::Entity** (p. 845).*

^ virtual **StatusCondition**^ **get_statuscondition** () override

*Allows access to the **DDS::StatusCondition** (p. 1183) associated with the **DDS::Entity** (p. 845).*

^ virtual **StatusMask get_status_changes** () override

*Retrieves the list of communication statuses in the **DDS::Entity** (p. 845) that are triggered.*

^ virtual **InstanceHandle_t get_instance_handle** () override

*Allows access to the **DDS::InstanceHandle_t** (p. 905) associated with the **DDS::Entity** (p. 845).*

Properties

^ static **DataReaderQos**^ **DATAREADER_QOS_DEFAULT** [get]

Special value for creating data reader with default QoS.

^ static **DataReaderQos**^ **DATAREADER_QOS_USE_TOPIC_QOS** [get]

*Special value for creating **DDS::DataReader** (p. 433) with a combination of the default **DDS::DataReaderQos** (p. 480) and the **DDS::TopicQos** (p. 1280).*

6.190.1 Detailed Description

<<***interface***>> *(p. 175)* A subscriber is the object responsible for actually receiving data from a subscription.

QoS:

DDS::SubscriberQos (p. 1230)

Status:

DDS::StatusKind::DATA_ON_READERS_STATUS

Listener:

DDS::SubscriberListener (p. 1226)

A subscriber acts on the behalf of one or several **DDS::DataReader** (p. 433) 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. 433) 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. 433) objects through the operation **get_datareaders()** (p. 1218) and then access the data available through operations on the **DDS::DataReader** (p. 433).

The following operations may be called even if the **DDS::Subscriber** (p. 1201) is not enabled. Other operations will the value **DDS::Retcode_NotEnabled** (p. 1121) if called on a disabled **DDS::Subscriber** (p. 1201):

- The base-class operations **DDS::Subscriber::set_qos** (p. 1221), **DDS::Subscriber::set_qos_with_profile** (p. 1221), **DDS::Subscriber::get_qos** (p. 1222), **DDS::Subscriber::set_listener** (p. 1222), **DDS::Subscriber::get_listener** (p. 1223), **DDS::Entity::enable** (p. 848), **DDS::Entity::get_statuscondition** (p. 849), **DDS::Entity::get_status_changes** (p. 850)

- **DDS::Subscriber::create_datareader** (p. 1210), **DDS::Subscriber::create_datareader_with_profile** (p. 1213), **DDS::Subscriber::delete_datareader** (p. 1214), **DDS::Subscriber::delete_contained_entities** (p. 1215), **DDS::Subscriber::set_default_datareader_qos** (p. 1206), **DDS::Subscriber::set_default_datareader_qos_with_profile** (p. 1207), **DDS::Subscriber::get_default_datareader_qos** (p. 1205), **DDS::Subscriber::set_default_library** (p. 1208), **DDS::Subscriber::set_default_profile** (p. 1209)

All operations except for the base-class operations **set_qos()** (p. 1221), **set_qos_with_profile()** (p. 1221), **get_qos()** (p. 1222), **set_listener()** (p. 1222), **get_listener()** (p. 1223), **enable()** (p. 1223) and **create_datareader()** (p. 1210) may fail with **DDS::Retcode_NotEnabled** (p. 1121).

See also:

Operations Allowed in Listener Callbacks (p. 954)

Examples:

HelloWorld_subscriber.cpp.

6.190.2 Member Function Documentation

6.190.2.1 void DDS::Subscriber::get_default_datareader_qos (DataReaderQos^ *qos*)

Copies the default **DDS::DataReaderQos** (p. 480) values into the provided **DDS::DataReaderQos** (p. 480) instance.

The retrieved `qos` will match the set of values specified on the last successful call to **DDS::Subscriber::set_default_datareader_qos** (p. 1206), or **DDS::Subscriber::set_default_datareader_qos_with_profile** (p. 1207), or else, if the call was never made, the default values from is owning **DDS::DomainParticipant** (p. 577).

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

MT Safety:

UNSAFE. It is not safe to retrieve the default QoS value from a subscriber while another thread may be simultaneously calling **DDS::Subscriber::set_default_datareader_qos** (p. 1206)

Parameters:

qos *<<inout>>* *(p. 176)* **DDS::DataReaderQos** (p. 480) to be filled-up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::Subscriber::DATAREADER_QOS_DEFAULT (p. 95)
DDS::Subscriber::create_datareader (p. 1210)

6.190.2.2 void DDS::Subscriber::set_default_datareader_qos (DataReaderQos^ *qos*)

Sets the default **DDS::DataReaderQos** (p. 480) values for this subscriber.

This call causes the default values inherited from the owning **DDS::DomainParticipant** (p. 577) to be overridden.

This default value will be used for newly created **DDS::DataReader** (p. 433) if **DDS::Subscriber::DATAREADER_QOS_DEFAULT** (p. 95) is specified as the `qos` parameter when **DDS::Subscriber::create_datareader** (p. 1210) is called.

Precondition:

The specified QoS policies must be consistent, or else the operation will have no effect and fail with **DDS::Retcode_InconsistentPolicy** (p. 1119)

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. 1206), **DDS::Subscriber::get_default_datareader_qos** (p. 1205) or calling **DDS::Subscriber::create_datareader** (p. 1210) with **DDS::Subscriber::DATAREADER_QOS_DEFAULT** (p. 95) as the `qos` parameter.

Parameters:

qos *<<in>> (p. 175)* The default **DDS::DataReaderQos** (p. 480) to be set to. The special value **DDS::Subscriber::DATAREADER_QOS_DEFAULT** (p. 95) may be passed as `qos` to indicate that the default QoS should be reset back to the initial values the factory would used if **DDS::Subscriber::set_default_datareader_qos** (p. 1206) had never been called. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), or or **DDS::Retcode_InconsistentPolicy** (p. 1119)

6.190.2.3 void DDS::Subscriber::set_default_datareader_qos_with_profile (System::String^ *library_name*, System::String^ *profile_name*)

<<eXtension>> (p. 174) Set the default **DDS::DataReaderQos** (p. 480) values for this subscriber based on the input XML QoS profile.

This default value will be used for newly created **DDS::DataReader** (p. 433) if **DDS::Subscriber::DATAREADER_QOS_DEFAULT** (p. 95) is specified as the `qos` parameter when **DDS::Subscriber::create_datareader** (p. 1210) is called.

Precondition:

The **DDS::DataReaderQos** (p. 480) contained in the specified XML QoS profile must be consistent, or else the operation will have no effect and fail with **DDS::Retcode_InconsistentPolicy** (p. 1119)

MT Safety:

UNSAFE. It is not safe to set the default QoS value from a **DDS::Subscriber** (p. 1201) while another thread may be simultaneously calling **DDS::Subscriber::set_default_datareader_qos** (p. 1206), **DDS::Subscriber::get_default_datareader_qos** (p. 1205) or calling **DDS::Subscriber::create_datareader** (p. 1210) with **DDS::Subscriber::DATAREADER_QOS_DEFAULT** (p. 95) as the `qos` parameter.

Parameters:

library_name *<<in>> (p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::Subscriber::set_default_library** (p. 1208)).

profile_name *<<**in**>>* *(p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::Subscriber::set_default_profile** (p. 1209)).

If the input profile cannot be found the method fails with **DDS::Retcode_-Error** (p. 1116).

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_-InconsistentPolicy** (p. 1119)

See also:

DDS::Subscriber::DATAREADER_QOS_DEFAULT (p. 95)
DDS::Subscriber::create_datareader_with_profile (p. 1213)

6.190.2.4 void DDS::Subscriber::set_default_library (System::String^ *library_name*)

*<<**eXtension**>>* *(p. 174)* Sets the default XML library for a **DDS::Subscriber** (p. 1201).

This method specifies the library that will be used as the default the next time a default library is needed during a call to one of this Subscriber's operations.

Any API requiring a library_name as a parameter can use null to refer to the default library.

If the default library is not set, the **DDS::Subscriber** (p. 1201) inherits the default from the **DDS::DomainParticipant** (p. 577) (see **DDS::DomainParticipant::set_default_library** (p. 596)).

Parameters:

library_name *<<**in**>>* *(p. 175)* Library name. If library_name is null any previous default is unset.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::Subscriber::get_default_library (p. 1209)

6.190.2.5 System::String ^ DDS::Subscriber::get_default_library ()

<<eXtension>> (p. 174) Gets the default XML library associated with a **DDS::Subscriber** (p. 1201).

Returns:

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

See also:

DDS::Subscriber::set_default_library (p. 1208)

6.190.2.6 void DDS::Subscriber::set_default_profile (System::String^ *library_name*, System::String^ *profile_name*)

<<eXtension>> (p. 174) Sets the default XML profile for a **DDS::Subscriber** (p. 1201).

This method specifies the profile that will be used as the default the next time a default **Subscriber** (p. 1201) profile is needed during a call to one of this Subscribers operations. When calling a **DDS::Subscriber** (p. 1201) method that requires a `profile_name` parameter, you can use NULL to refer to the default profile. (This same information applies to setting a default library.)

If the default profile is not set, the **DDS::Subscriber** (p. 1201) inherits the default from the **DDS::DomainParticipant** (p. 577) (see **DDS::DomainParticipant::set_default_profile** (p. 597)).

This method does not set the default QoS for **DDS::DataReader** (p. 433) objects created by this **DDS::Subscriber** (p. 1201); for this functionality, use **DDS::Subscriber::set_default_datareader_qos_with_profile** (p. 1207) (you may pass in NULL after having called **set_default_profile()** (p. 1209)).

This method does not set the default QoS for newly created Subscribers; for this functionality, use **DDS::DomainParticipant::set_default_subscriber_qos_with_profile** (p. 614).

Parameters:

library_name *<<in>> (p. 175)* The library name containing the profile.

profile_name *<<in>> (p. 175)* The profile name. If profile_name is null any previous default is unset.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::Subscriber::get_default_profile (p. 1210)
DDS::Subscriber::get_default_profile_library (p. 1210)

6.190.2.7 System::String ^ DDS::Subscriber::get_default_profile ()

<<eXtension>> (p. 174) Gets the default XML profile associated with a **DDS::Subscriber** (p. 1201).

Returns:

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

See also:

DDS::Subscriber::set_default_profile (p. 1209)

6.190.2.8 System::String ^ DDS::Subscriber::get_default_profile_library ()

<<eXtension>> (p. 174) Gets the library where the default XML QoS profile is contained for a **DDS::Subscriber** (p. 1201).

The default profile library is automatically set when **DDS::Subscriber::set_default_profile** (p. 1209) is called.

This library can be different than the **DDS::Subscriber** (p. 1201) default library (see **DDS::Subscriber::get_default_library** (p. 1209)).

Returns:

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

See also:

DDS::Subscriber::set_default_profile (p. 1209)

6.190.2.9 DataReader ^ DDS::Subscriber::create_datareader (ITopicDescription^ *topic*, DataReaderQos^ *qos*, DataReaderListener^ *listener*, StatusMask *mask*)

Creates a **DDS::DataReader** (p. 433) that will be attached and belong to the **DDS::Subscriber** (p. 1201).

For each application-defined type `Foo` (p. 877), there is an implied, auto-generated class `DDS::TypedDataReader` (p. 1338) (an incarnation of **Foo-DataReader** (p. 878)) that extends **DDS::DataReader** (p. 433) and contains the operations to read data of type `Foo` (p. 877).

Note that a common application pattern to construct the QoS for the **DDS::DataReader** (p. 433) is to:

- Retrieve the QoS policies on the associated **DDS::Topic** (p. 1258) by means of the **DDS::Topic::get_qos** (p. 1262) operation.
- Retrieve the default **DDS::DataReader** (p. 433) qos by means of the **DDS::Subscriber::get_default_datareader_qos** (p. 1205) operation.
- Combine those two QoS policies (for example, using **DDS::Subscriber::copy_from_topic_qos** (p. 1220)) and selectively modify policies as desired
- Use the resulting QoS policies to construct the **DDS::DataReader** (p. 433).

When a **DDS::DataReader** (p. 433) is created, only those transports already registered are available to the **DDS::DataReader** (p. 433). See **Built-in Transport Plugins** (p. 122) for details on when a builtin transport is registered.

MT Safety:

UNSAFE. If **DDS::Subscriber::DATAREADER_QOS_DEFAULT** (p. 95) 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. 1206).

Precondition:

If subscriber is enabled, the topic must be enabled. If it is not, this operation will fail and no **DDS::DataReader** (p. 433) will be created.
The given DDS::TopicDescription must have been created from the same participant as this subscriber. If it was created from a different participant, this method will return NULL.
If `qos` is DDS::DATAREADER_QOS_USE_TOPIC_QOS, `topic` cannot be **DDS::MultiTopic** (p. 984), or else this method will return NULL.

Parameters:

topic <<*in*>> *(p. 175)* The DDS::TopicDescription that the **DDS::DataReader** (p. 433) will be associated with. Cannot be NULL.

qos *<<in>>* *(p. 175)* The qos of the **DDS::DataReader** (p. 433). The special value **DDS::Subscriber::DATAREADER_QOS_DEFAULT** (p. 95) can be used to indicate that the **DDS::DataReader** (p. 433) should be created with the default **DDS::DataReaderQos** (p. 480) set in the **DDS::Subscriber** (p. 1201). If DDS::TopicDescription is of type **DDS::Topic** (p. 1258) or **DDS::ContentFilteredTopic** (p. 419), the special value DDS::DATAREADER_QOS_USE_TOPIC_QOS can be used to indicate that the **DDS::DataReader** (p. 433) should be created with the combination of the default **DDS::DataReaderQos** (p. 480) set on the **DDS::Subscriber** (p. 1201) and the **DDS::TopicQos** (p. 1280) (in the case of a **DDS::ContentFilteredTopic** (p. 419), the **DDS::TopicQos** (p. 1280) of the related **DDS::Topic** (p. 1258)). if DDS::DATAREADER_QOS_USE_TOPIC_QOS is used, `topic` cannot be a **DDS::MultiTopic** (p. 984). Cannot be NULL.

listener *<<in>>* *(p. 175)* The listener of the **DDS::DataReader** (p. 433).

mask *<<in>>* *(p. 175)*. Changes of communication status to be invoked on the listener.

Returns:

A **DDS::DataReader** (p. 433) of a derived class specific to the data-type associated with the **DDS::Topic** (p. 1258) or NULL if an error occurred.

See also:

DDS::TypedDataReader (p. 1338)
Specifying QoS on entities (p. 267) for information on setting QoS before entity creation
DDS::DataReaderQos (p. 480) for rules on consistency among QoS
DDS::Subscriber::create_datareader_with_profile (p. 1213)
DDS::Subscriber::get_default_datareader_qos (p. 1205)
DDS::Topic::set_qos (p. 1261)
DDS::Subscriber::copy_from_topic_qos (p. 1220)
DDS::DataReader::set_listener (p. 450)

Examples:

HelloWorld_subscriber.cpp.

6.190.2.10 DataReader ^ DDS::Subscriber::create_datareader_-with_profile (ITopicDescription^ *topic*, System::String^ *library_name*, System::String^ *profile_name*, DataReaderListener^ *listener*, StatusMask *mask*)

<<eXtension>> (p. 174) Creates a **DDS::DataReader** (p. 433) object using the **DDS::DataReaderQos** (p. 480) associated with the input XML QoS profile.

The **DDS::DataReader** (p. 433) will be attached and belong to the **DDS::Subscriber** (p. 1201).

For each application-defined type `Foo` (p. 877), there is an implied, auto-generated class `DDS::TypedDataReader` (p. 1338) (an incarnation of **Foo-DataReader** (p. 878)) that extends **DDS::DataReader** (p. 433) and contains the operations to read data of type `Foo` (p. 877).

When a **DDS::DataReader** (p. 433) is created, only those transports already registered are available to the **DDS::DataReader** (p. 433). See **Built-in Transport Plugins** (p. 122) 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. 433) will be created.
The given DDS::TopicDescription must have been created from the same participant as this subscriber. If it was created from a different participant, this method will return NULL.

Parameters:

topic *<<in>> (p. 175)* The DDS::TopicDescription that the **DDS::DataReader** (p. 433) will be associated with. Cannot be NULL.

library_name *<<in>> (p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::Subscriber::set_default_library** (p. 1208)).

profile_name *<<in>> (p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::Subscriber::set_default_profile** (p. 1209)).

listener *<<in>> (p. 175)* The listener of the **DDS::DataReader** (p. 433).

mask *<<in>> (p. 175)*. Changes of communication status to be invoked on the listener.

Returns:

A **DDS::DataReader** (p. 433) of a derived class specific to the data-type associated with the **DDS::Topic** (p. 1258) or NULL if an error occurred.

See also:

DDS::TypedDataReader (p. 1338)
Specifying QoS on entities (p. 267) for information on setting QoS before entity creation
DDS::DataReaderQos (p. 480) for rules on consistency among QoS
DDS::Subscriber::DATAREADER_QOS_DEFAULT (p. 95)
DDS::DATAREADER_QOS_USE_TOPIC_QOS
DDS::Subscriber::create_datareader (p. 1210)
DDS::Subscriber::get_default_datareader_qos (p. 1205)
DDS::Topic::set_qos (p. 1261)
DDS::Subscriber::copy_from_topic_qos (p. 1220)
DDS::DataReader::set_listener (p. 450)

6.190.2.11 void DDS::Subscriber::delete_datareader (DataReader^ % *a_datareader*)

Deletes a **DDS::DataReader** (p. 433) that belongs to the **DDS::Subscriber** (p. 1201).

Precondition:

If the **DDS::DataReader** (p. 433) does not belong to the **DDS::Subscriber** (p. 1201), or if ther are any existing **DDS::ReadCondition** (p. 1084) or **DDS::QueryCondition** (p. 1082) objects that are attached to the **DDS::DataReader** (p. 433), 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_PreconditionNotMet** (p. 1123).

Postcondition:

Listener (p. 952) installed on the **DDS::DataReader** (p. 433) will not be called after this method completes successfully.

Parameters:

a_datareader *<<in>>* *(p. 175)* The **DDS::DataReader** (p. 433) to be deleted.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_PreconditionNotMet** (p. 1123).

6.190.2.12 void DDS::Subscriber::delete_contained_entities ()

Deletes all the entities that were created by means of the "create" operation on the **DDS::Subscriber** (p. 1201).

Deletes all contained **DDS::DataReader** (p. 433) objects. This pattern is applied recursively. In this manner, the operation **DDS::Subscriber::delete_contained_entities** (p. 1215) on the **DDS::Subscriber** (p. 1201) will end up deleting all the entities recursively contained in the **DDS::Subscriber** (p. 1201), that is also the **DDS::QueryCondition** (p. 1082) and **DDS::ReadCondition** (p. 1084) objects belonging to the contained **DDS::DataReader** (p. 433).

The operation will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123) 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. 433) cannot be deleted because the application has called a **DDS::TypedDataReader::read** (p. 1341) or **DDS::TypedDataReader::take** (p. 1342) operation and has not called the corresponding **DDS::TypedDataReader::return_loan** (p. 1364) operation to return the loaned samples.

Once **DDS::Subscriber::delete_contained_entities** (p. 1215) completes successfully, the application may delete the **DDS::Subscriber** (p. 1201), knowing that it has no contained **DDS::DataReader** (p. 433) objects.

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_PreconditionNotMet** (p. 1123)

6.190.2.13 DataReader ^ DDS::Subscriber::lookup_datareader (System::String^ *topic_name*)

Retrieves an existing **DDS::DataReader** (p. 433).

Use this operation on the built-in **DDS::Subscriber** (p. 1201) (**Built-in Topics** (p. 42)) to access the built-in **DDS::DataReader** (p. 433) entities for the built-in topics.

The built-in **DDS::DataReader** (p. 433) is created when this operation is called on a built-in topic for the first time. The built-in **DDS::DataReader** (p. 433) is deleted automatically when the **DDS::DomainParticipant** (p. 577) is deleted.

To ensure that builtin **DDS::DataReader** (p. 433) entities receive all the discovery traffic, it is suggested that you lookup the builtin **DDS::DataReader** (p. 433) before the **DDS::DomainParticipant** (p. 577) is enabled. Looking up builtin **DDS::DataReader** (p. 433) may implicitly register builtin transports

due to creation of **DDS::DataReader** (p. 433) (see **Built-in Transport Plugins** (p. 122) 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. 577).
- (optional) Modify builtin transport properties
- Call **DDS::DomainParticipant::get_builtin_subscriber()** (p. 632).
- Call **lookup_datareader()** (p. 1215).
- Call **enable()** (p. 1223) on the **DomainParticipant** (p. 577).

Parameters:

topic_name *<<in>>* *(p. 175)* Name of the DDS::TopicDescription that the retrieved **DDS::DataReader** (p. 433) is attached to. Cannot be NULL.

Returns:

A **DDS::DataReader** (p. 433) that belongs to the **DDS::Subscriber** (p. 1201) attached to the DDS::TopicDescription with `topic_name`. If no such **DDS::DataReader** (p. 433) exists, this operation returns NULL.

The returned **DDS::DataReader** (p. 433) may be enabled or disabled.

If more than one **DDS::DataReader** (p. 433) is attached to the **DDS::Subscriber** (p. 1201), this operation may return any one of them.

MT Safety:

UNSAFE. It is not safe to lookup a **DDS::DataReader** (p. 433) in one thread while another thread is simultaneously creating or destroying that **DDS::DataReader** (p. 433).

6.190.2.14 void DDS::Subscriber::begin_access ()

[**Not supported (optional)**] Indicates that the application is about to access the data samples in any of the **DDS::DataReader** (p. 433) objects attached to the **DDS::Subscriber** (p. 1201).

The application is **required** to use this operation only if the **PRESENTATION** (p. 279) policy of the **DDS::Subscriber**

(p. 1201) to which a **DDS::DataReader** (p. 433) belongs has the **DDS::PresentationQosPolicy::access_scope** (p. 1015) set to DDS::PresentationQosPolicyAccessScopeKind::GROUP_PRESENTATION_QOS.

In the aforementioned case, the operation **begin_access()** (p. 1216) must be called prior to calling any of the sample-accessing operations, namely: **get_datareaders()** (p. 1218) on the **DDS::Subscriber** (p. 1201) and **DDS::TypedDataReader::read** (p. 1341), **DDS::TypedDataReader::take** (p. 1342), **DDS::TypedDataReader::read_w_condition** (p. 1349), **DDS::TypedDataReader::take_w_condition** (p. 1350) on any **DDS::DataReader** (p. 433). Otherwise the sample-accessing operations will fail with the error **DDS::Retcode_PreconditionNotMet** (p. 1123).

Once the application has finished accessing the data samples it must call **end_access()** (p. 1217).

The application is not required to call **begin_access()** (p. 1216) / **end_access()** (p. 1217) if the **PRESENTATION** (p. 279) policy has the **DDS::PresentationQosPolicy::access_scope** (p. 1015) set to something other than DDS::PresentationQosPolicyAccessScopeKind::GROUP_PRESENTATION_QOS.

Calling **begin_access()** (p. 1216) / **end_access()** (p. 1217) in this case is not considered an error and has no effect.

Calls to **begin_access()** (p. 1216) / **end_access()** (p. 1217) may be nested and must be balanced.

Exceptions:

DDS::Retcode_Unsupported *(p. **1125**)*

6.190.2.15 void DDS::Subscriber::end_access ()

[**Not supported (optional)**] Indicates that the application has finished accessing the data samples in **DDS::DataReader** (p. 433) objects managed by the **DDS::Subscriber** (p. 1201).

This operation must be used to close a corresponding **begin_access()** (p. 1216).

After calling **end_access()** (p. 1217) the application should no longer access any of the `Data` or **DDS::SampleInfo** (p. 1148) elements returned from the sample-accessing operations. This call must close a previous call to **begin_access()** (p. 1216), otherwise the operation will fail with the error **DDS::Retcode_PreconditionNotMet** (p. 1123).

Exceptions:

DDS::Retcode_Unsupported (p. 1125)

6.190.2.16 void DDS::Subscriber::get_datareaders (DataReaderSeq^ *readers*, System::UInt32 *sample_states*, System::UInt32 *view_states*, System::UInt32 *instance_states*)

Allows the application to access the **DDS::DataReader** (p. 433) objects that contain samples with the specified `sample_states`, `view_states` and `instance_states`.

If the **PRESENTATION** (p. 279) policy of the **DDS::Subscriber** (p. 1201) to which the **DDS::DataReader** (p. 433) belongs has the **DDS::PresentationQosPolicy::access_scope** (p. 1015) set to DDS::PresentationQosPolicyAccessScopeKind::GROUP_PRESENTATION_QOS, this operation should only be invoked inside a **begin_access()** (p. 1216) / **end_access()** (p. 1217) block. Otherwise, it will fail with the error **DDS::Retcode_PreconditionNotMet** (p. 1123).

Depending on the setting of the **PRESENTATION** (p. 279) policy, the returned collection of **DDS::DataReader** (p. 433) objects may be a 'set' containing each **DDS::DataReader** (p. 433) at most once in no specified order, or a 'list' containing each **DDS::DataReader** (p. 433) one or more times in a specific order.

- If **DDS::PresentationQosPolicy::access_scope** (p. 1015) is idref_PresentationQosPolicyAccessScopeKind_INSTANCE_PRESENTATION_QOS or idref_PresentationQosPolicyAccessScopeKind_TOPIC_PRESENTATION_QOS, the returned collection is a 'set.'

- If **DDS::PresentationQosPolicy::access_scope** (p. 1015) is DDS::PresentationQosPolicyAccessScopeKind::GROUP_PRESENTATION_QOS and **DDS::PresentationQosPolicy::ordered_access** (p. 1016) is set to true, then the returned collection is a 'list'.

This difference is due to the fact that, in the second situation it is required to access samples belonging to different **DDS::DataReader** (p. 433) objects in a particular order. In this case, the application should process each **DDS::DataReader** (p. 433) in the same order it appears in the list and `read()` or `take()` exactly one sample from each **DDS::DataReader** (p. 433). The patterns that an application should use to access data is fully described in **Access to data samples** (p. 91)

Parameters:

readers *<<inout>>* *(p. 176)* a **DDS::DataReaderSeq** (p. 498) object where the set or list of readers will be returned. Cannot be NULL.

sample_states *<<in>>* *(p. 175)* the returned data reader must contain samples that have one of these `sample_states`.

view_states *<<in>>* *(p. 175)* the returned data reader must contain samples that have one of these `view_states`.

instance_states *<<in>>* *(p. 175)* the returned data reader must contain samples that have one of these `instance_states`.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

Access to data samples (p. 91)
PRESENTATION (p. 279)

6.190.2.17 void DDS::Subscriber::notify_datareaders ()

Invokes the operation **DDS::DataReaderListener::on_data_available()** (p. 463) on the **DDS::DataReaderListener** (p. 461) objects attached to contained **DDS::DataReader** (p. 433) entities with DDS::StatusKind::DATA_AVAILABLE_STATUS that is considered changed as described in **Changes in read communication status** (p. 241).

This operation is typically invoked from the **DDS::SubscriberListener::on_data_on_readers** (p. 1229) operation in the **DDS::SubscriberListener** (p. 1226). That way the **DDS::SubscriberListener** (p. 1226) can delegate to the **DDS::DataReaderListener** (p. 461) objects the handling of the data.

The operation will notify the data readers that have a `sample_state` of **DDS::SampleStateKind::NOT_READ_SAMPLE_STATE** (p. 1162), `view_state` of **DDS::SampleStateKind::ANY_SAMPLE_STATE** (p. 103) and `instance_state` of **DDS::InstanceStateKind::ANY_INSTANCE_STATE** (p. 105).

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_NotEnabled** (p. 1121).

6.190.2.18 DomainParticipant ^ DDS::Subscriber::get_participant ()

Returns the **DDS::DomainParticipant** (p. 577) to which the **DDS::Subscriber** (p. 1201) belongs.

Returns:

the **DDS::DomainParticipant** (p. 577) to which the **DDS::Subscriber** (p. 1201) belongs.

6.190.2.19 void DDS::Subscriber::copy_from_topic_qos (DataReaderQos^ *datareader_qos*, TopicQos^ *topic_qos*)

Copies the policies in the **DDS::TopicQos** (p. 1280) to the corresponding policies in the **DDS::DataReaderQos** (p. 480).

Copies the policies in the **DDS::TopicQos** (p. 1280) to the corresponding policies in the **DDS::DataReaderQos** (p. 480) (replacing values in the **DDS::DataReaderQos** (p. 480), if present).

This is a "convenience" operation most useful in combination with the operations **DDS::Subscriber::get_default_datareader_qos** (p. 1205) and **DDS::Topic::get_qos** (p. 1262). The operation **DDS::Subscriber::copy_from_topic_qos** (p. 1220) can be used to merge the **DDS::DataReader** (p. 433) default QoS policies with the corresponding ones on the **DDS::Topic** (p. 1258). The resulting QoS can then be used to create a new **DDS::DataReader** (p. 433), or set its QoS.

This operation does not check the resulting **DDS::DataReaderQos** (p. 480) for consistency. This is because the 'merged' **DDS::DataReaderQos** (p. 480) may not be the final one, as the application can still modify some policies prior to applying the policies to the **DDS::DataReader** (p. 433).

Parameters:

datareader_qos *<<inout>> (p. 176)* **DDS::DataReaderQos** (p. 480) to be filled-up. Cannot be NULL.

topic_qos *<<in>> (p. 175)* **DDS::TopicQos** (p. 1280) to be merged with **DDS::DataReaderQos** (p. 480). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.190.2.20 void DDS::Subscriber::set_qos (SubscriberQos^ *qos*)

Sets the subscriber QoS.

This operation modifies the QoS of the **DDS::Subscriber** (p. 1201).

The **DDS::SubscriberQos::group_data** (p. 1231), **DDS::SubscriberQos::partition** (p. 1231) and **DDS::SubscriberQos::entity_factory** (p. 1231) can be changed. The other policies are immutable.

Parameters:

qos *<<in>>* *(p. 175)* **DDS::SubscriberQos** (p. 1230) to be set to. Policies must be consistent. Immutable policies cannot be changed after **DDS::Subscriber** (p. 1201) is enabled. The special value **DDS::DomainParticipant::SUBSCRIBER_QOS_DEFAULT** (p. 38) can be used to indicate that the QoS of the **DDS::Subscriber** (p. 1201) should be changed to match the current default **DDS::SubscriberQos** (p. 1230) set in the **DDS::DomainParticipant** (p. 577). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_ImmutablePolicy** (p. 1118), or **DDS::Retcode_InconsistentPolicy** (p. 1119).

See also:

DDS::SubscriberQos (p. 1230) for rules on consistency among QoS
set_qos (abstract) (p. 846)
Operations Allowed in Listener Callbacks (p. 954)

6.190.2.21 void DDS::Subscriber::set_qos_with_profile (System::String^ *library_name*, System::String^ *profile_name*)

<<eXtension>> *(p. 174)* Change the QoS of this subscriber using the input XML QoS profile.

This operation modifies the QoS of the **DDS::Subscriber** (p. 1201).

The **DDS::SubscriberQos::group_data** (p. 1231), **DDS::SubscriberQos::partition** (p. 1231) and **DDS::SubscriberQos::entity_factory** (p. 1231) can be changed. The other policies are immutable.

Parameters:

library_name <<***in***>> *(*p. *175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::Subscriber::set_default_library** (p. 1208)).

profile_name <<***in***>> *(*p. *175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::Subscriber::set_default_profile** (p. 1209)).

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_ImmutablePolicy** (p. 1118), or **DDS::Retcode_InconsistentPolicy** (p. 1119).

See also:

DDS::SubscriberQos (p. 1230) for rules on consistency among QoS
Operations Allowed in Listener Callbacks (p. 954)

6.190.2.22 void DDS::Subscriber::get_qos (SubscriberQos^ *qos*)

Gets the subscriber QoS.

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

Parameters:

qos <<***in***>> *(*p. *175)* **DDS::SubscriberQos** (p. 1230) to be filled in. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

get_qos (abstract) (p. 847)

6.190.2.23 void DDS::Subscriber::set_listener (SubscriberListener^ *l*, StatusMask *mask*)

Sets the subscriber listener.

Parameters:

l *<<in>>* *(p. 175)* **DDS::SubscriberListener** (p. 1226) to set to.

mask *<<in>>* *(p. 175)* DDS::StatusMask associated with the **DDS::SubscriberListener** (p. 1226).

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

set_listener (abstract) (p. 847)

6.190.2.24 SubscriberListener ^ DDS::Subscriber::get_listener ()

Get the subscriber listener.

Returns:

DDS::SubscriberListener (p. 1226) of the **DDS::Subscriber** (p. 1201).

See also:

get_listener (abstract) (p. 848)

6.190.2.25 virtual void DDS::Subscriber::enable () [override, virtual]

Enables the **DDS::Entity** (p. 845).

This operation enables the **Entity** (p. 845). **Entity** (p. 845) objects can be created either enabled or disabled. This is controlled by the value of the **ENTITY_FACTORY** (p. 304) QoS policy on the corresponding factory for the **DDS::Entity** (p. 845).

By default, **ENTITY_FACTORY** (p. 304) is set so that it is not necessary to explicitly call **DDS::Entity::enable** (p. 848) on newly created entities.

The **DDS::Entity::enable** (p. 848) operation is idempotent. Calling enable on an already enabled **Entity** (p. 845) returns OK and has no effect.

If a **DDS::Entity** (p. 845) 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. 849)
- 'factory' operations
- **DDS::Entity::get_status_changes** (p. 850) 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_NotEnabled** (p. 1121).

It is legal to delete an **DDS::Entity** (p. 845) that has not been enabled by calling the proper operation on its factory.

Entities created from a factory that is disabled are created disabled, regardless of the setting of the **DDS::EntityFactoryQosPolicy** (p. 851).

Calling enable on an **Entity** (p. 845) whose factory is not enabled will fail and return **DDS::Retcode_PreconditionNotMet** (p. 1123).

If **DDS::EntityFactoryQosPolicy::autoenable_created_entities** (p. 852) is TRUE, the enable operation on a factory will automatically enable all entities created from that factory.

Listeners associated with an entity are not called until the entity is enabled.

Conditions associated with a disabled entity are "inactive," that is, they have a `trigger_value` == FALSE.

Exceptions:

One of the **Standard Return Codes** (p. 235), **Standard Return Codes** (p. 235) or **DDS::Retcode_PreconditionNotMet** (p. 1123).

Implements **DDS::Entity** (p. 848).

6.190.2.26 virtual StatusCondition ^ DDS::Subscriber::get_statuscondition () `[override, virtual]`

Allows access to the **DDS::StatusCondition** (p. 1183) associated with the **DDS::Entity** (p. 845).

The returned condition can then be added to a **DDS::WaitSet** (p. 1411) so that the application can wait for specific status changes that affect the **DDS::Entity** (p. 845).

Returns:

the status condition associated with this entity.

Implements **DDS::Entity** (p. 849).

6.190.2.27 virtual StatusMask DDS::Subscriber::get_status_changes () [override, virtual]

Retrieves the list of communication statuses in the **DDS::Entity** (p. 845) that are triggered.

That is, the list of statuses whose value has changed since the last time the application read the status using the get_*_status() method.

When the entity is first created or if the entity is not enabled, all communication statuses are in the "untriggered" state so the list returned by the get_status_changes operation will be empty.

The list of statuses returned by the get_status_changes operation refers to the status that are triggered on the **Entity** (p. 845) itself and does not include statuses that apply to contained entities.

Returns:

list of communication statuses in the **DDS::Entity** (p. 845) that are triggered.

See also:

Status Kinds (p. 238)

Implements **DDS::Entity** (p. 850).

6.190.2.28 virtual InstanceHandle_t DDS::Subscriber::get_instance_handle () [override, virtual]

Allows access to the **DDS::InstanceHandle_t** (p. 905) associated with the **DDS::Entity** (p. 845).

This operation returns the **DDS::InstanceHandle_t** (p. 905) that represents the **DDS::Entity** (p. 845).

Returns:

the instance handle associated with this entity.

Implements **DDS::Entity** (p. 850).

6.191 DDS::SubscriberListener Class Reference

<<*interface*>> *(p. 175)* **DDS::Listener** (p. 952) for status about a subscriber.

`#include <managed_subscription.h>`

Inheritance diagram for DDS::SubscriberListener::

Public Member Functions

- ^ virtual void **on_requested_deadline_missed** (**DataReader**^ reader, **RequestedDeadlineMissedStatus**% status)

  Handles the DDS::StatusKind::REQUESTED_DEADLINE_MISSED_STATUS communication status.

- ^ virtual void **on_liveliness_changed** (**DataReader**^ reader, **LivelinessChangedStatus**% status)

  Handles the DDS::StatusKind::LIVELINESS_CHANGED_STATUS communication status.

- ^ virtual void **on_requested_incompatible_qos** (**DataReader**^ reader, **RequestedIncompatibleQosStatus**^ status)

  Handles the DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS communication status.

- ^ virtual void **on_sample_rejected** (**DataReader**^ reader, **SampleRejectedStatus**% status)

  Handles the DDS::StatusKind::SAMPLE_REJECTED_STATUS communication status.

- ^ virtual void **on_data_available** (**DataReader**^ reader)

  Handle the DDS::StatusKind::DATA_AVAILABLE_STATUS communication status.

- ^ virtual void **on_sample_lost** (**DataReader**^ reader, **SampleLostStatus**% status)

  Handles the DDS::StatusKind::SAMPLE_LOST_STATUS communication status.

- ^ virtual void **on_subscription_matched** (**DataReader**^ reader, **SubscriptionMatchedStatus**% status)

Handles the DDS::StatusKind::SUBSCRIPTION_MATCHED_STATUS communication status.

^ virtual void **on_data_on_readers** (**Subscriber**^ sub)

Handles the DDS::StatusKind::DATA_ON_READERS_STATUS communication status.

6.191.1 Detailed Description

<<***interface***>> *(p. 175)* **DDS::Listener** (p. 952) for status about a subscriber.

Entity:

DDS::Subscriber (p. 1201)

Status:

DDS::StatusKind::DATA_AVAILABLE_STATUS;
DDS::StatusKind::DATA_ON_READERS_STATUS;
DDS::StatusKind::LIVELINESS_CHANGED_STATUS, **DDS::LivelinessChangedStatus** (p. 956);
DDS::StatusKind::REQUESTED_DEADLINE_MISSED_STATUS, **DDS::RequestedDeadlineMissedStatus** (p. 1105);
DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS, **DDS::RequestedIncompatibleQosStatus** (p. 1107);
DDS::StatusKind::SAMPLE_LOST_STATUS, **DDS::SampleLostStatus** (p. 1158);
DDS::StatusKind::SAMPLE_REJECTED_STATUS, **DDS::SampleRejectedStatus** (p. 1159);
DDS::StatusKind::SUBSCRIPTION_MATCHED_STATUS, **DDS::SubscriptionMatchedStatus** (p. 1244)

See also:

DDS::Listener (p. 952)
Status Kinds (p. 238)
Operations Allowed in Listener Callbacks (p. 954)

6.191.2 Member Function Documentation

6.191.2.1 virtual void DDS::SubscriberListener::on_requested_deadline_missed (DataReader^ *reader*, RequestedDeadlineMissedStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::REQUESTED_DEADLINE_MISSED_STATUS communication status.

Reimplemented from **DDS::DataReaderListener** (p. 462).

Reimplemented in **DDS::DomainParticipantListener** (p. 681).

6.191.2.2 virtual void DDS::SubscriberListener::on_liveliness_changed (DataReader^ *reader*, LivelinessChangedStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::LIVELINESS_CHANGED_STATUS communication status.

Reimplemented from **DDS::DataReaderListener** (p. 463).

Reimplemented in **DDS::DomainParticipantListener** (p. 681).

6.191.2.3 virtual void DDS::SubscriberListener::on_requested_incompatible_qos (DataReader^ *reader*, RequestedIncompatibleQosStatus^ *status*) [inline, virtual]

Handles the DDS::StatusKind::REQUESTED_INCOMPATIBLE_QOS_STATUS communication status.

Reimplemented from **DDS::DataReaderListener** (p. 463).

Reimplemented in **DDS::DomainParticipantListener** (p. 681).

6.191.2.4 virtual void DDS::SubscriberListener::on_sample_rejected (DataReader^ *reader*, SampleRejectedStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::SAMPLE_REJECTED_STATUS communication status.

Reimplemented from **DDS::DataReaderListener** (p. 463).

Reimplemented in **DDS::DomainParticipantListener** (p. 681).

6.191.2.5 virtual void DDS::SubscriberListener::on_data_available (DataReader^ *reader*) [inline, virtual]

Handle the DDS::StatusKind::DATA_AVAILABLE_STATUS communication status.

Reimplemented from **DDS::DataReaderListener** (p. 463).

Reimplemented in **DDS::DomainParticipantListener** (p. 682).

6.191.2.6 virtual void DDS::SubscriberListener::on_sample_lost (DataReader^ *reader*, SampleLostStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::SAMPLE_LOST_STATUS communication status.

Reimplemented from **DDS::DataReaderListener** (p. 464).

Reimplemented in **DDS::DomainParticipantListener** (p. 682).

6.191.2.7 virtual void DDS::SubscriberListener::on_subscription_matched (DataReader^ *reader*, SubscriptionMatchedStatus% *status*) [inline, virtual]

Handles the DDS::StatusKind::SUBSCRIPTION_MATCHED_STATUS communication status.

Reimplemented from **DDS::DataReaderListener** (p. 464).

Reimplemented in **DDS::DomainParticipantListener** (p. 682).

6.191.2.8 virtual void DDS::SubscriberListener::on_data_on_readers (Subscriber^ *sub*) [inline, virtual]

Handles the DDS::StatusKind::DATA_ON_READERS_STATUS communication status.

Reimplemented in **DDS::DomainParticipantListener** (p. 682).

6.192 DDS::SubscriberQos Class Reference

QoS policies supported by a **DDS::Subscriber** (p. 1201) entity.

`#include <managed_subscription.h>`

Public Attributes

- **PresentationQosPolicy presentation**

  Presentation policy, ***PRESENTATION*** *(p. 279).*

- **PartitionQosPolicy^ partition**

  Partition policy, ***PARTITION*** *(p. 289).*

- **GroupDataQosPolicy^ group_data**

  Group data policy, ***GROUP_DATA*** *(p. 275).*

- **EntityFactoryQosPolicy entity_factory**

  Entity *(p. 845) factory policy,* ***ENTITY_FACTORY*** *(p. 304).*

- **ExclusiveAreaQosPolicy exclusive_area**

  <<**eXtension**>> (p. 174) *Exclusive area for the subscriber and all entities that are created by the subscriber.*

6.192.1 Detailed Description

QoS policies supported by a **DDS::Subscriber** (p. 1201) entity.

You must set certain members in a consistent manner:

length of DDS::SubscriberQos::group_data.value <= DDS::DomainParticipantQos::resource_limits.subscriber_group_data_max_length

length of DDS::SubscriberQos::partition.name <= DDS::DomainParticipantQos::resource_limits.max_partitions

combined number of characters (including terminating 0) in DDS::SubscriberQos::partition.name <= DDS::DomainParticipantQos::resource_limits.max_partition_cumulative_characters

If any of the above are not true, **DDS::Subscriber::set_qos** (p. 1221) and **DDS::Subscriber::set_qos_with_profile** (p. 1221) will fail with **DDS::Retcode_InconsistentPolicy** (p. 1119)

6.192.2 Member Data Documentation

6.192.2.1 PresentationQosPolicy DDS::SubscriberQos::presentation

Presentation policy, **PRESENTATION** (p. 279).

6.192.2.2 PartitionQosPolicy ^ DDS::SubscriberQos::partition

Partition policy, **PARTITION** (p. 289).

6.192.2.3 GroupDataQosPolicy ^ DDS::SubscriberQos::group_data

Group data policy, **GROUP_DATA** (p. 275).

6.192.2.4 EntityFactoryQosPolicy DDS::SubscriberQos::entity_factory

Entity (p. 845) factory policy, **ENTITY_FACTORY** (p. 304).

6.192.2.5 ExclusiveAreaQosPolicy DDS::SubscriberQos::exclusive_area

<<***eXtension***>> *(p. 174)* Exclusive area for the subscriber and all entities that are created by the subscriber.

6.193 DDS::SubscriberSeq Class Reference

Declares IDL `sequence` < **DDS::Subscriber** (p. 1201) > .

`#include <managed_subscription.h>`

Inheritance diagram for DDS::SubscriberSeq::

6.193.1 Detailed Description

Declares IDL `sequence` < **DDS::Subscriber** (p. 1201) > .

See also:

DDS::Sequence (p. 1163)

6.194 DDS::SubscriptionBuiltinTopicData Class Reference

Entry created when a **DDS::DataReader** (p. 433) is discovered in association with its **Subscriber** (p. 1201).

`#include <managed_builtin.h>`

Inheritance diagram for DDS::SubscriptionBuiltinTopicData::

Public Attributes

- ^ **BuiltinTopicKey_t key**

  DCPS key to distinguish entries.

- ^ **BuiltinTopicKey_t participant_key**

  *DCPS key of the participant to which the **DataReader** (p. 433) belongs.*

- ^ System::String^ **topic_name**

  *Name of the related **DDS::Topic** (p. 1258).*

- ^ System::String^ **type_name**

  *Name of the type attached to the **DDS::Topic** (p. 1258).*

- ^ **DurabilityQosPolicy durability**

  *Policy of the corresponding **DataReader** (p. 433).*

- ^ **DeadlineQosPolicy deadline**

  *Policy of the corresponding **DataReader** (p. 433).*

- ^ **LatencyBudgetQosPolicy latency_budget**

  *Policy of the corresponding **DataReader** (p. 433).*

- ^ **LivelinessQosPolicy liveliness**

  *Policy of the corresponding **DataReader** (p. 433).*

- ^ **ReliabilityQosPolicy reliability**

  *Policy of the corresponding **DataReader** (p. 433).*

- ^ **OwnershipQosPolicy ownership**

  *Policy of the corresponding **DataReader** (p. 433).*

- **DestinationOrderQosPolicy destination_order**

  *Policy of the corresponding **DataReader** (p. 433).*

- **TimeBasedFilterQosPolicy time_based_filter**

  *Policy of the corresponding **DataReader** (p. 433).*

- **PresentationQosPolicy presentation**

  *Policy of the **Subscriber** (p. 1201) to which the **DataReader** (p. 433) belongs.*

- **BuiltinTopicKey_t subscriber_key**

  <<**eXtension**>> (p. 174) *DCPS key of the subscriber to which the **DataReader** (p. 433) belongs.*

- **GUID_t virtual_guid**

  <<**eXtension**>> (p. 174) *Virtual GUID associated to the **DataReader** (p. 433).*

- **ProtocolVersion_t rtps_protocol_version**

  <<**eXtension**>> (p. 174) *Version number of the RTPS wire protocol used.*

- **VendorId_t rtps_vendor_id**

  <<**eXtension**>> (p. 174) *ID of vendor implementing the RTPS wire protocol.*

- **ProductVersion_t product_version**

  <<**eXtension**>> (p. 174) *This is a vendor specific parameter. It gives the current version for rti-dds.*

Properties

- **UserDataQosPolicy^ user_data** `[get]`

  *Policy of the corresponding **DataReader** (p. 433).*

- **PartitionQosPolicy^ partition** `[get]`

  *Policy of the **Subscriber** (p. 1201) to which the **DataReader** (p. 433) belongs.*

- **TopicDataQosPolicy^ topic_data** `[get]`

  *Policy of the related **Topic** (p. 1258).*

^ **GroupDataQosPolicy^ group_data** [get]

*Policy of the **Subscriber** (p. 1201) to which the **DataReader** (p. 433) belongs.*

^ **DDS::TypeCode^ type_code** [get]

<<**eXtension**>> (p. 174) *Type code information of the corresponding **Topic** (p. 1258)*

^ **PropertyQosPolicy^ property_qos** [get]

<<**eXtension**>> (p. 174) *Properties of the corresponding **DataReader** (p. 433).*

^ **LocatorSeq^ unicast_locators** [get]

<<**eXtension**>> (p. 174) *Custom unicast locators that the endpoint can specify. The default locators will be used if this is not specified.*

^ **LocatorSeq^ multicast_locators** [get]

<<**eXtension**>> (p. 174) *Custom multicast locators that the endpoint can specify. The default locators will be used if this is not specified.*

^ **ContentFilterProperty_t^ content_filter_property** [get]

<<**eXtension**>> (p. 174) *This field provides all the required information to enable content filtering on the Writer side.*

^ System::Boolean **disable_positive_acks** [get, set]

<<**eXtension**>> (p. 174) *This is a vendor specific parameter. Determines whether the corresponding **DataReader** (p. 433) sends positive acknowledgements for reliability.*

6.194.1 Detailed Description

Entry created when a **DDS::DataReader** (p. 433) is discovered in association with its **Subscriber** (p. 1201).

Data associated with the built-in topic **DDS::SubscriptionBuiltinTopicDataTypeSupport::SUBSCRIPTION_TOPIC_NAME** (p. 232). It contains QoS policies and additional information that apply to the remote **DDS::DataReader** (p. 433) the related **DDS::Subscriber** (p. 1201).

See also:

DDS::SubscriptionBuiltinTopicDataTypeSupport::SUBSCRIPTION_TOPIC_NAME (p. 232)
DDS::SubscriptionBuiltinTopicDataDataReader (p. 1241)

6.194.2 Member Data Documentation

6.194.2.1 BuiltinTopicKey_t DDS::SubscriptionBuiltinTopicData::key

DCPS key to distinguish entries.

6.194.2.2 BuiltinTopicKey_t DDS::SubscriptionBuiltinTopicData::participant_key

DCPS key of the participant to which the **DataReader** (p. 433) belongs.

6.194.2.3 System::String ^ DDS::SubscriptionBuiltinTopicData::topic_name

Name of the related **DDS::Topic** (p. 1258).

The length of this string is limited to 255 characters.

6.194.2.4 System::String ^ DDS::SubscriptionBuiltinTopicData::type_name

Name of the type attached to the **DDS::Topic** (p. 1258).

The length of this string is limited to 255 characters.

6.194.2.5 DurabilityQosPolicy DDS::SubscriptionBuiltinTopicData::durability

Policy of the corresponding **DataReader** (p. 433).

6.194.2.6 DeadlineQosPolicy DDS::SubscriptionBuiltinTopicData::deadline

Policy of the corresponding **DataReader** (p. 433).

6.194.2.7 LatencyBudgetQosPolicy DDS::SubscriptionBuiltinTopicData::latency_budget

Policy of the corresponding **DataReader** (p. 433).

6.194.2.8 LivelinessQosPolicy DDS::SubscriptionBuiltinTopicData::liveliness

Policy of the corresponding **DataReader** (p. 433).

6.194.2.9 ReliabilityQosPolicy DDS::SubscriptionBuiltinTopicData::reliability

Policy of the corresponding **DataReader** (p. 433).

6.194.2.10 OwnershipQosPolicy DDS::SubscriptionBuiltinTopicData::ownership

Policy of the corresponding **DataReader** (p. 433).

6.194.2.11 DestinationOrderQosPolicy DDS::SubscriptionBuiltinTopicData::destination_order

Policy of the corresponding **DataReader** (p. 433).

6.194.2.12 TimeBasedFilterQosPolicy DDS::SubscriptionBuiltinTopicData::time_based_filter

Policy of the corresponding **DataReader** (p. 433).

6.194.2.13 PresentationQosPolicy DDS::SubscriptionBuiltinTopicData::presentation

Policy of the **Subscriber** (p. 1201) to which the **DataReader** (p. 433) belongs.

6.194.2.14 BuiltinTopicKey_t DDS::SubscriptionBuiltinTopicData::subscriber_key

<<*eXtension*>> *(p. 174)* DCPS key of the subscriber to which the **DataReader** (p. 433) belongs.

6.194.2.15 GUID_t DDS::SubscriptionBuiltinTopicData::virtual_guid

<<*eXtension*>> *(p. 174)* Virtual GUID associated to the **DataReader** (p. 433).

See also:

DDS::GUID_t (p. 894)

6.194.2.16 ProtocolVersion_t DDS::SubscriptionBuiltinTopicData::rtps_protocol_version

*<<**eXtension**>> (p. 174)* Version number of the RTPS wire protocol used.

6.194.2.17 VendorId_t DDS::SubscriptionBuiltinTopicData::rtps_vendor_id

*<<**eXtension**>> (p. 174)* ID of vendor implementing the RTPS wire protocol.

6.194.2.18 ProductVersion_t DDS::SubscriptionBuiltinTopicData::product_version

*<<**eXtension**>> (p. 174)* This is a vendor specific parameter. It gives the current version for rti-dds.

6.194.3 Property Documentation

6.194.3.1 UserDataQosPolicy^ DDS::SubscriptionBuiltinTopicData::user_data [get]

Policy of the corresponding **DataReader** (p. 433).

6.194.3.2 PartitionQosPolicy^ DDS::SubscriptionBuiltinTopicData::partition [get]

Policy of the **Subscriber** (p. 1201) to which the **DataReader** (p. 433) belongs.

6.194.3.3 TopicDataQosPolicy^ DDS::SubscriptionBuiltinTopicData::topic_data [get]

Policy of the related **Topic** (p. 1258).

6.194.3.4 GroupDataQosPolicy^ DDS::SubscriptionBuiltinTopicData::group_data [get]

Policy of the **Subscriber** (p. 1201) to which the **DataReader** (p. 433) belongs.

6.194.3.5 DDS:: TypeCode^ DDS::SubscriptionBuiltinTopicData::type_code [get]

<<*eXtension*>> *(p. 174)* Type code information of the corresponding **Topic** (p. 1258)

6.194.3.6 PropertyQosPolicy^ DDS::SubscriptionBuiltinTopicData::property_qos [get]

<<*eXtension*>> *(p. 174)* Properties of the corresponding **DataReader** (p. 433).

6.194.3.7 LocatorSeq^ DDS::SubscriptionBuiltinTopicData::unicast_locators [get]

<<*eXtension*>> *(p. 174)* Custom unicast locators that the endpoint can specify. The default locators will be used if this is not specified.

6.194.3.8 LocatorSeq^ DDS::SubscriptionBuiltinTopicData::multicast_locators [get]

<<*eXtension*>> *(p. 174)* Custom multicast locators that the endpoint can specify. The default locators will be used if this is not specified.

6.194.3.9 ContentFilterProperty_t^ DDS::SubscriptionBuiltinTopicData::content_filter_property [get]

<<*eXtension*>> *(p. 174)* This field provides all the required information to enable content filtering on the Writer side.

6.194.3.10 System:: Boolean DDS::SubscriptionBuiltinTopicData::disable_positive_acks [get, set]

<<*eXtension*>> *(p. 174)* This is a vendor specific parameter. Determines whether the corresponding **DataReader** (p. 433) sends positive acknowledgements for reliability.

6.195 DDS::SubscriptionBuiltinTopicDataDataReader Class Reference

Instantiates `DataReader` (p. 433) < **DDS::SubscriptionBuiltinTopicData** (p. 1233) > .

`#include <managed_builtin.h>`

Inheritance diagram for DDS::SubscriptionBuiltinTopicDataDataReader::

6.195.1 Detailed Description

Instantiates `DataReader` (p. 433) < **DDS::SubscriptionBuiltinTopicData** (p. 1233) > .

DDS::DataReader (p. 433) of topic **DDS::SubscriptionBuiltinTopicDataTypeSupport::SUBSCRIPTION_TOPIC_NAME** (p. 232) used for accessing **DDS::SubscriptionBuiltinTopicData** (p. 1233) of the remote **DDS::DataReader** (p. 433) and the associated **DDS::Subscriber** (p. 1201).

Instantiates:

<<*generic*>> *(p. 175)* **DDS::TypedDataReader** (p. 1338)

See also:

DDS::SubscriptionBuiltinTopicData (p. 1233)
DDS::SubscriptionBuiltinTopicDataTypeSupport::SUBSCRIPTION_TOPIC_NAME (p. 232)

6.196 DDS::SubscriptionBuiltinTopicDataSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < **DDS::SubscriptionBuiltinTopicData** (p. 1233) > .

`#include <managed_builtin.h>`

6.196.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < **DDS::SubscriptionBuiltinTopicData** (p. 1233) > .

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

DDS::SubscriptionBuiltinTopicData (p. 1233)

6.197 DDS::SubscriptionBuiltinTopicDataTypeSupport Class Reference

Instantiates `TypeSupport` (p. 1385) < **DDS::SubscriptionBuiltinTopicData** (p. 1233) > .

`#include <managed_builtin.h>`

Inherits DDS::AbstractBuiltinTopicDataTypeSupport< T >.

Properties

^ static System::String^ **SUBSCRIPTION_TOPIC_NAME** `[get]`

Subscription topic name.

6.197.1 Detailed Description

Instantiates `TypeSupport` (p. 1385) < **DDS::SubscriptionBuiltinTopicData** (p. 1233) > .

Instantiates:

<<***generic***>> *(p. 175)* **FooTypeSupport** (p. 884)

See also:

DDS::SubscriptionBuiltinTopicData (p. 1233)

6.198 DDS::SubscriptionMatchedStatus Struct Reference

DDS::StatusKind::SUBSCRIPTION_MATCHED_STATUS

`#include <managed_subscription.h>`

Public Attributes

- System::Int32 **total_count**

  *The total cumulative number of times the concerned **DDS::DataReader** (p. 433) discovered a "match" with a **DDS::DataWriter** (p. 499).*

- System::Int32 **total_count_change**

  The change in total_count since the last time the listener was called or the status was read.

- System::Int32 **current_count**

  *The current number of writers with which the **DDS::DataReader** (p. 433) is matched.*

- System::Int32 **current_count_peak**

  <<**eXtension**>> (p. 174) *The highest value that current_count has reached until now.*

- System::Int32 **current_count_change**

  The change in current_count since the last time the listener was called or the status was read.

- **InstanceHandle_t last_publication_handle**

  *A handle to the last **DDS::DataWriter** (p. 499) that caused the status to change.*

6.198.1 Detailed Description

DDS::StatusKind::SUBSCRIPTION_MATCHED_STATUS

A "match" happens when the **DDS::DataReader** (p. 433) finds a **DDS::DataWriter** (p. 499) for the same **DDS::Topic** (p. 1258) with an offered QoS that is compatible with that requested by the **DDS::DataReader** (p. 433).

This status is also changed (and the listener, if any, called) when a match is ended. A local **DDS::DataReader** (p. 433) will become "unmatched" from

a remote **DDS::DataWriter** (p. 499) when that **DDS::DataWriter** (p. 499) goes away for any reason.

Examples:

HelloWorld_subscriber.cpp.

6.198.2 Member Data Documentation

6.198.2.1 System::Int32 DDS::SubscriptionMatchedStatus::total_count

The total cumulative number of times the concerned **DDS::DataReader** (p. 433) discovered a "match" with a **DDS::DataWriter** (p. 499).

This number increases whenever a new match is discovered. It does not change when an existing match goes away.

6.198.2.2 System::Int32 DDS::SubscriptionMatchedStatus::total_count_change

The change in total_count since the last time the listener was called or the status was read.

6.198.2.3 System::Int32 DDS::SubscriptionMatchedStatus::current_count

The current number of writers with which the **DDS::DataReader** (p. 433) is matched.

This number increases when a new match is discovered and decreases when an existing match goes away.

6.198.2.4 System::Int32 DDS::SubscriptionMatchedStatus::current_count_peak

<<***eXtension***>> *(p. 174)* The highest value that current_count has reached until now.

6.198.2.5 System::Int32 DDS::SubscriptionMatchedStatus::current_count_change

The change in current_count since the last time the listener was called or the status was read.

6.198.2.6 InstanceHandle_t DDS::SubscriptionMatchedStatus::last_publication_handle

A handle to the last **DDS::DataWriter** (p. 499) that caused the status to change.

6.199 DDS::SystemResourceLimitsQosPolicy Struct Reference

Configures **DDS::DomainParticipant** (p. 577)-independent resources used by RTI Data Distribution Service. Mainly used to change the maximum number of **DDS::DomainParticipant** (p. 577) entities that can be created within a single process (address space).

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_systemresourcelimits_qos_policy_name** ()

  *Stringified human-readable name for **DDS::SystemResourceLimitsQosPolicy** (p. 1247).*

Public Attributes

- System::Int32 **max_objects_per_thread**

  *The maximum number of objects that can be stored per thread for a **DDS::DomainParticipantFactory** (p. 649).*

6.199.1 Detailed Description

Configures **DDS::DomainParticipant** (p. 577)-independent resources used by RTI Data Distribution Service. Mainly used to change the maximum number of **DDS::DomainParticipant** (p. 577) entities that can be created within a single process (address space).

This QoS policy is an extension to the DDS standard.

Entity:

DDS::DomainParticipantFactory (p. 649)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)

6.199.2 Usage

Within a single process (or address space for some supported real-time operating systems), applications may create and use multiple **DDS::DomainParticipant** (p. 577) entities. This QoS policy sets a parameter that places an effective upper bound on the maximum number of **DDS::DomainParticipant** (p. 577) entities that can be created in a single process/address space.

6.199.3 Member Data Documentation

6.199.3.1 System::Int32 DDS::SystemResourceLimitsQosPolicy::max_objects_per_thread

The maximum number of objects that can be stored per thread for a **DDS::DomainParticipantFactory** (p. 649).

Before increasing this value to allow you to create more participants, carefully consider the application design that requires you to create so many participants. Remember: a **DDS::DomainParticipant** (p. 577) is a heavy-weight object. It spawns several threads and maintains its own discovery database (see **DISCOVERY** (p. 320)). Creating more participants than RTI Data Distribution Service strictly requires – one per domain per process/address space – can adversely affect the performance and resource utilization of your application.

[default] 1024; this value allows you to create about 10 or 11 **DDS::DomainParticipant** (p. 577) entities.

[range] [1, 1 billion]

6.200 DDS::ThreadSettings_t Class Reference

The properties of a thread of execution.

`#include <managed_infrastructure.h>`

Public Attributes

- ^ **ThreadSettingsKind mask**

  Describes the type of thread.

- ^ System::Int32 **priority**

  Thread priority.

- ^ System::Int32 **stack_size**

  The thread stack-size.

- ^ **IntSeq^ cpu_list**

  The list of processors on which the thread(s) may run.

- ^ **ThreadSettingsCpuRotationKind cpu_rotation**

  Determines how processor affinity is applied to multiple threads.

6.200.1 Detailed Description

The properties of a thread of execution.

QoS:

DDS::EventQosPolicy (p. 858) **DDS::DatabaseQosPolicy** (p. 428) **DDS::ReceiverPoolQosPolicy** (p. 1090) **DDS::AsynchronousPublisherQosPolicy** (p. 371)

6.200.2 Member Data Documentation

6.200.2.1 ThreadSettingsKind DDS::ThreadSettings_t::mask

Describes the type of thread.

[**default**] 0, use default options of the OS

6.200.2.2 System::Int32 DDS::ThreadSettings_t::priority

Thread priority.

[range] Platform-dependent

6.200.2.3 System::Int32 DDS::ThreadSettings_t::stack_size

The thread stack-size.

[range] Platform-dependent.

6.200.2.4 IntSeq ^ DDS::ThreadSettings_t::cpu_list

The list of processors on which the thread(s) may run.

A sequence of integers that represent the set of processors on which the thread(s) controlled by this QoS may run. An empty sequence (the default) means the middleware wlll make no CPU affinity adjustments.

Note: This feature is currently only supported on a subset of architectures (see the `Platform Notes`). The API may change as more architectures are added in future releases.

This value is only relevant to the **DDS::ReceiverPoolQosPolicy** (p. 1090). It is ignored within other QoS policies that include **DDS::ThreadSettings_t** (p. 1249).

See also:

Controlling CPU Core Affinity for RTI Threads (p. 258)

[default] Empty sequence

6.200.2.5 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. 1090). It is ignored within other QoS policies that include **DDS::ThreadSettings_t** (p. 1249).

See also:

Controlling CPU Core Affinity for RTI Threads (p. 258)

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.;

6.201 DDS::Time_t Struct Reference

Type for *time* representation.

`#include <managed_infrastructure.h>`

Public Member Functions

- System::Boolean **is_zero** ()
  Check if time is zero.
- System::Boolean **is_invalid_time** ()

Public Attributes

- System::Int32 **sec**
  seconds
- System::UInt32 **nanosec**
  nanoseconds

Properties

- static System::Int32 **TIME_INVALID_SEC** `[get]`
  A sentinel indicating an invalid second of time.
- static System::Int32 **TIME_INVALID_NSEC** `[get]`
  A sentinel indicating an invalid nano-second of time.
- static **Time_t TIME_ZERO** `[get]`
  The default instant in time: zero seconds and zero nanoseconds.
- static **Time_t TIME_INVALID** `[get]`
  A sentinel indicating an invalid time.

6.201.1 Detailed Description

Type for *time* representation.

A **DDS::Time_t** (p. 1252) represents a moment in time.

6.201.2 Member Data Documentation

6.201.2.1 System::Int32 DDS::Time_t::sec

seconds

6.201.2.2 System::UInt32 DDS::Time_t::nanosec

nanoseconds

6.202 DDS::TimeBasedFilterQosPolicy Struct Reference

Filter that allows a **DDS::DataReader** (p. 433) to specify that it is interested only in (potentially) a subset of the values of the data.

`#include <managed_infrastructure.h>`

Static Public Member Functions

^ static System::String^ **get_timebasedfilter_qos_policy_name** ()

*Stringified human-readable name for **DDS::TimeBasedFilterQosPolicy** (p. 1254).*

Public Attributes

^ **Duration_t minimum_separation**

The minimum separation duration between subequent samples.

6.202.1 Detailed Description

Filter that allows a **DDS::DataReader** (p. 433) 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. 433) does not want to receive more than one value each `minimum_separation`, regardless of how fast the changes occur.

Entity:

DDS::DataReader (p. 433)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **YES** (p. 269)

6.202.2 Usage

You can use this QoS policy to reduce the amount of data received by a **DDS::DataReader** (p. 433). **DDS::DataWriter** (p. 499) entities may send

data faster than needed by a **DDS::DataReader** (p. 433). For example, a **DDS::DataReader** (p. 433) 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 measure in tenths (0.1) of a second up to several seconds. However, a **DDS::DataWriter** (p. 499) of sensor information may have other **DDS::DataReader** (p. 433) 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. 433) entities can set their own time-based filters, so that data published faster than the period set by a each **DDS::DataReader** (p. 433) will not be delivered to that **DDS::DataReader** (p. 433).

The **TIME_BASED_FILTER** (p. 288) also applies to each instance separately; that is, the constraint is that the **DDS::DataReader** (p. 433) 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. 433), 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, unicast data delivery, if the data type is unkeyed and the **DDS::DataWriter** (p. 499) has an infinite **DDS::LivelinessQosPolicy::lease_duration** (p. 963), RTI Data Distribution Service will only send as many packets to a **DDS::DataReader** (p. 433) as required by the TIME_BASED_FILTER, no matter how fast **DDS::TypedDataWriter::write** (p. 1376) is called.

However, in configurations where RTI Data Distribution Service must send all the data published by the **DDS::DataWriter** (p. 499) (for example, when the **DDS::DataWriter** (p. 499) is reliable, when the data type is keyed, or when the **DDS::DataWriter** (p. 499) has a finite **DDS::LivelinessQosPolicy::lease_duration** (p. 963)), only the data that passes the TIME_BASED_FILTER will be stored in the receive queue of the **DDS::DataReader** (p. 433). 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).

6.202.3 Consistency

It is inconsistent for a **DDS::DataReader** (p. 433) to have a `minimum_separation` longer than its **DEADLINE** (p. 281) `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. 557) for more information about the interactions between deadlines and time-based filters.

The setting of a **TIME_BASED_FILTER** (p. 288) – that is, the selection of a `minimum_separation` with a value greater than zero – is consistent with all settings of the **HISTORY** (p. 294) and **RELIABILITY** (p. 290) QoS. The **TIME_BASED_FILTER** (p. 288) specifies the samples that are of interest to the **DDS::DataReader** (p. 433). The **HISTORY** (p. 294) and **RELIABILITY** (p. 290) 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. 433); that is, they apply *after* the **TIME_BASED_FILTER** (p. 288) has been applied.

In the case where the reliability QoS kind is DDS::ReliabilityQosPolicyKind::RELIABLE_RELIABILITY_QOS, in steady-state – defined as the situation where the **DDS::DataWriter** (p. 499) 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. 433).

See also:

DeadlineQosPolicy (p. 557)
HistoryQosPolicy (p. 898)
ReliabilityQosPolicy (p. 1094)

6.202.4 Member Data Documentation

6.202.4.1 Duration_t DDS::TimeBasedFilterQosPolicy::minimum_separation

The minimum separation duration between subequent samples.

[**default**] 0 (meaning the **DDS::DataReader** (p. 433) is potentially interested in all values)

[**range**] [0,1 year], < **DDS::DeadlineQosPolicy::period** (p. 559)

6.203 DDS::Topic Class Reference

*<<**interface**>> (p. 175)* The most basic description of the data to be published and subscribed.

`#include <managed_topic.h>`

Inheritance diagram for DDS::Topic::

Public Member Functions

- ^ void **get_inconsistent_topic_status** (**InconsistentTopicStatus**% status)

  *Allows the application to retrieve the DDS::StatusKind::INCONSISTENT_TOPIC_STATUS status of a **DDS::Topic** (p. 1258).*

- ^ void **set_qos** (**TopicQos**^ qos)

  Set the topic QoS.

- ^ void **set_qos_with_profile** (System::String^ library_name, System::String^ profile_name)

  <<**eXtension**>> (p. 174) *Change the QoS of this topic using the input XML QoS profile.*

- ^ void **get_qos** (**TopicQos**^ qos)

  Get the topic QoS.

- ^ void **set_listener** (**TopicListener**^ l, **StatusMask** mask)

  Set the topic listener.

- ^ **TopicListener**^ **get_listener** ()

  Get the topic listener.

- ^ virtual System::String^ **get_type_name** ()

  Get the associated `type_name`.

- ^ virtual System::String^ **get_name** ()

  Get the name used to create this DDS::TopicDescription .

- ^ virtual **DomainParticipant**^ **get_participant** ()

  *Get the **DDS::DomainParticipant** (p. 577) to which the DDS::TopicDescription belongs.*

^ virtual void **enable** () override

*Enables the **DDS::Entity** (p. 845).*

^ virtual **StatusCondition^ get_statuscondition** () override

*Allows access to the **DDS::StatusCondition** (p. 1183) associated with the **DDS::Entity** (p. 845).*

^ virtual **StatusMask get_status_changes** () override

*Retrieves the list of communication statuses in the **DDS::Entity** (p. 845) that are triggered.*

^ virtual **InstanceHandle_t get_instance_handle** () override

*Allows access to the **DDS::InstanceHandle_t** (p. 905) associated with the **DDS::Entity** (p. 845).*

Static Public Member Functions

^ static **Topic^ narrow** (**ITopicDescription^** topic_description)

*Narrow the given DDS::TopicDescription pointer to a **DDS::Topic** (p. 1258) pointer.*

6.203.1 Detailed Description

*<<**interface**>> (p. 175)* The most basic description of the data to be published and subscribed.

QoS:

DDS::TopicQos (p. 1280)

Status:

DDS::StatusKind::INCONSISTENT_TOPIC_STATUS,
DDS::InconsistentTopicStatus (p. 903)

Listener:

DDS::TopicListener (p. 1278)

A **DDS::Topic** (p. 1258) is identified by its name, which must be unique in the whole domain. In addition (by virtue of extending DDS::TopicDescription) it

fully specifies the type of the data that can be communicated when publishing or subscribing to the **DDS::Topic** (p. 1258).

DDS::Topic (p. 1258) is the only DDS::TopicDescription that can be used for publications and therefore associated with a **DDS::DataWriter** (p. 499).

The following operations may be called even if the **DDS::Topic** (p. 1258) is not enabled. Other operations will fail with the value **DDS::Retcode_NotEnabled** (p. 1121) if called on a disabled **DDS::Topic** (p. 1258):

- All the base-class operations **set_qos()** (p. 1261), **set_qos_with_profile()** (p. 1262), **get_qos()** (p. 1262), **set_listener()** (p. 1263), **get_listener()** (p. 1263) **enable()** (p. 1265), **get_statuscondition()** (p. 1266) and **get_status_changes()** (p. 1266)
- **get_inconsistent_topic_status()** (p. 1260)

See also:

Operations Allowed in Listener Callbacks (p. 954)

Examples:

HelloWorld_publisher.cpp, and **HelloWorld_subscriber.cpp**.

6.203.2 Member Function Documentation

6.203.2.1 static Topic ^ DDS::Topic::narrow (ITopicDescription^ *topic_description*) [static]

Narrow the given DDS::TopicDescription pointer to a **DDS::Topic** (p. 1258) pointer.

Returns:

DDS::Topic (p. 1258) if this DDS::TopicDescription is a **DDS::Topic** (p. 1258). Otherwise, return NULL.

6.203.2.2 void DDS::Topic::get_inconsistent_topic_status (InconsistentTopicStatus% *status*)

Allows the application to retrieve the DDS::StatusKind::INCONSISTENT_TOPIC_STATUS status of a **DDS::Topic** (p. 1258).

Retrieve the current **DDS::InconsistentTopicStatus** (p. 903)

Parameters:

status *<<inout>> (p. 176)* Status to be retrieved. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

DDS::InconsistentTopicStatus (p. 903)

6.203.2.3 void DDS::Topic::set_qos (TopicQos^ *qos*)

Set the topic QoS.

The **DDS::TopicQos::topic_data** (p. 1281) and **DDS::TopicQos::deadline** (p. 1282), **DDS::TopicQos::latency_budget** (p. 1282), **DDS::TopicQos::transport_priority** (p. 1282) and **DDS::TopicQos::lifespan** (p. 1282) can be changed. The other policies are immutable.

Parameters:

qos *<<in>> (p. 175)* Set of policies to be applied to **DDS::Topic** (p. 1258).

Policies must be consistent. Immutable policies cannot be changed after **DDS::Topic** (p. 1258) is enabled. The special value **DDS::DomainParticipant::TOPIC_QOS_DEFAULT** (p. 39) can be used to indicate that the QoS of the **DDS::Topic** (p. 1258) should be changed to match the current default **DDS::TopicQos** (p. 1280) set in the **DDS::DomainParticipant** (p. 577). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_ImmutablePolicy** (p. 1118) if immutable policy is changed, or **DDS::Retcode_InconsistentPolicy** (p. 1119) if policies are inconsistent

See also:

DDS::TopicQos (p. 1280) for rules on consistency among QoS
set_qos (abstract) (p. 846)
Operations Allowed in Listener Callbacks (p. 954)

6.203.2.4 void DDS::Topic::set_qos_with_profile (System::String^ *library_name*, System::String^ *profile_name*)

<<eXtension>> (p. 174) Change the QoS of this topic using the input XML QoS profile.

The **DDS::TopicQos::topic_data** (p. 1281) and **DDS::TopicQos::deadline** (p. 1282), **DDS::TopicQos::latency_budget** (p. 1282), **DDS::TopicQos::transport_priority** (p. 1282) and **DDS::TopicQos::lifespan** (p. 1282) can be changed. The other policies are immutable.

Parameters:

library_name *<<in>> (p. 175)* Library name containing the XML QoS profile. If library_name is null RTI Data Distribution Service will use the default library (see **DDS::DomainParticipant::set_default_library** (p. 596)).

profile_name *<<in>> (p. 175)* XML QoS Profile name. If profile_name is null RTI Data Distribution Service will use the default profile (see **DDS::DomainParticipant::set_default_profile** (p. 597)).

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_ImmutablePolicy** (p. 1118) if immutable policy is changed, or **DDS::Retcode_InconsistentPolicy** (p. 1119) if policies are inconsistent

See also:

DDS::TopicQos (p. 1280) for rules on consistency among QoS
Operations Allowed in Listener Callbacks (p. 954)

6.203.2.5 void DDS::Topic::get_qos (TopicQos^ *qos*)

Get the topic QoS.

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

Parameters:

qos *<<inout>> (p. 176)* QoS to be filled up. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

get_qos (abstract) (p. 847)

6.203.2.6 void DDS::Topic::set_listener (TopicListener^ *l*, StatusMask *mask*)

Set the topic listener.

Parameters:

l <<*in*>> *(p. 175)* **Listener** (p. 952) to be installed on entity.

mask <<*in*>> *(p. 175)* Changes of communication status to be invoked on the listener.

Exceptions:

One of the **Standard Return Codes** (p. 235)

See also:

set_listener (abstract) (p. 847)

6.203.2.7 TopicListener ^ DDS::Topic::get_listener ()

Get the topic listener.

Returns:

Existing listener attached to the **DDS::Topic** (p. 1258).

See also:

get_listener (abstract) (p. 848)

6.203.2.8 virtual System::String ^ DDS::Topic::get_type_name () [virtual]

Get the associated `type_name`.

The type name defines a locally unique type for the publication or the subscription.

The `type_name` corresponds to a unique string used to register a type via the **FooTypeSupport::register_type** (p. 885) method.

Thus, the `type_name` implies an association with a corresponding **DDS::TypeSupport** (p. 1385) and this DDS::TopicDescription.

Returns:

the type name. The returned type name is valid until the DDS::TopicDescription is deleted.

Postcondition:

The result is non-NULL.

See also:

DDS::TypeSupport (p. 1385), **FooTypeSupport** (p. 884)

Implements **DDS::ITopicDescription** (p. 913).

6.203.2.9 virtual System::String ^ DDS::Topic::get_name () [virtual]

Get the name used to create this DDS::TopicDescription .

Returns:

the name used to create this DDS::TopicDescription. The returned topic name is valid until the DDS::TopicDescription is deleted.

Postcondition:

The result is non-NULL.

Implements **DDS::ITopicDescription** (p. 913).

6.203.2.10 virtual DomainParticipant ^ DDS::Topic::get_participant () [virtual]

Get the **DDS::DomainParticipant** (p. 577) to which the DDS::TopicDescription belongs.

Returns:

The **DDS::DomainParticipant** (p. 577) to which the DDS::TopicDescription belongs.

Postcondition:

The result is non-NULL.

Implements **DDS::ITopicDescription** (p. 914).

6.203.2.11 virtual void DDS::Topic::enable () [override, virtual]

Enables the **DDS::Entity** (p. 845).

This operation enables the **Entity** (p. 845). **Entity** (p. 845) objects can be created either enabled or disabled. This is controlled by the value of the **ENTITY_FACTORY** (p. 304) QoS policy on the corresponding factory for the **DDS::Entity** (p. 845).

By default, **ENTITY_FACTORY** (p. 304) is set so that it is not necessary to explicitly call **DDS::Entity::enable** (p. 848) on newly created entities.

The **DDS::Entity::enable** (p. 848) operation is idempotent. Calling enable on an already enabled **Entity** (p. 845) returns OK and has no effect.

If a **DDS::Entity** (p. 845) 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. 849)
- 'factory' operations
- **DDS::Entity::get_status_changes** (p. 850) 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_NotEnabled** (p. 1121).

It is legal to delete an **DDS::Entity** (p. 845) that has not been enabled by calling the proper operation on its factory.

Entities created from a factory that is disabled are created disabled, regardless of the setting of the **DDS::EntityFactoryQosPolicy** (p. 851).

Calling enable on an **Entity** (p. 845) whose factory is not enabled will fail and return **DDS::Retcode_PreconditionNotMet** (p. 1123).

If **DDS::EntityFactoryQosPolicy::autoenable_created_entities** (p. 852) is TRUE, the enable operation on a factory will automatically enable all entities created from that factory.

Listeners associated with an entity are not called until the entity is enabled.

Conditions associated with a disabled entity are "inactive," that is, they have a `trigger_value` == FALSE.

Exceptions:

One of the **Standard Return Codes** (p. 235), **Standard Return Codes** (p. 235) or **DDS::Retcode_PreconditionNotMet** (p. 1123).

Implements **DDS::Entity** (p. 848).

6.203.2.12 virtual StatusCondition ^ DDS::Topic::get_statuscondition () [override, virtual]

Allows access to the **DDS::StatusCondition** (p. 1183) associated with the **DDS::Entity** (p. 845).

The returned condition can then be added to a **DDS::WaitSet** (p. 1411) so that the application can wait for specific status changes that affect the **DDS::Entity** (p. 845).

Returns:

the status condition associated with this entity.

Implements **DDS::Entity** (p. 849).

6.203.2.13 virtual StatusMask DDS::Topic::get_status_changes () [override, virtual]

Retrieves the list of communication statuses in the **DDS::Entity** (p. 845) that are triggered.

That is, the list of statuses whose value has changed since the last time the application read the status using the get_*_status() method.

When the entity is first created or if the entity is not enabled, all communication statuses are in the "untriggered" state so the list returned by the get_status_changes operation will be empty.

The list of statuses returned by the get_status_changes operation refers to the status that are triggered on the **Entity** (p. 845) itself and does not include statuses that apply to contained entities.

Returns:

list of communication statuses in the **DDS::Entity** (p. 845) that are triggered.

See also:

Status Kinds (p. 238)

Implements **DDS::Entity** (p. 850).

6.203.2.14 virtual InstanceHandle_t DDS::Topic::get_instance_handle () [override, virtual]

Allows access to the **DDS::InstanceHandle_t** (p. 905) associated with the **DDS::Entity** (p. 845).

This operation returns the **DDS::InstanceHandle_t** (p. 905) that represents the **DDS::Entity** (p. 845).

Returns:

the instance handle associated with this entity.

Implements **DDS::Entity** (p. 850).

6.204 DDS::TopicBuiltinTopicData Class Reference

Entry created when a **Topic** (p. 1258) object discovered.

`#include <managed_builtin.h>`

Inheritance diagram for DDS::TopicBuiltinTopicData::

Public Attributes

- **BuiltinTopicKey_t key**
  DCPS key to distinguish entries.
- System::String^ **name**
  *Name of the **DDS::Topic** (p. 1258).*
- System::String^ **type_name**
  *Name of the type attached to the **DDS::Topic** (p. 1258).*
- **DurabilityQosPolicy durability**
  *durability policy of the corresponding **Topic** (p. 1258)*
- **DurabilityServiceQosPolicy durability_service**
  *durability service policy of the corresponding **Topic** (p. 1258)*
- **DeadlineQosPolicy deadline**
  *Policy of the corresponding **Topic** (p. 1258).*
- **LatencyBudgetQosPolicy latency_budget**
  *Policy of the corresponding **Topic** (p. 1258).*
- **LivelinessQosPolicy liveliness**
  *Policy of the corresponding **Topic** (p. 1258).*
- **ReliabilityQosPolicy reliability**
  *Policy of the corresponding **Topic** (p. 1258).*
- **TransportPriorityQosPolicy transport_priority**
  *Policy of the corresponding **Topic** (p. 1258).*

^ **LifespanQosPolicy lifespan**

Policy of the corresponding ***Topic*** *(p. 1258).*

^ **DestinationOrderQosPolicy destination_order**

Policy of the corresponding ***Topic*** *(p. 1258).*

^ **HistoryQosPolicy history**

Policy of the corresponding ***Topic*** *(p. 1258).*

^ **ResourceLimitsQosPolicy resource_limits**

Policy of the corresponding ***Topic*** *(p. 1258).*

^ **OwnershipQosPolicy ownership**

Policy of the corresponding ***Topic*** *(p. 1258).*

Properties

^ **TopicDataQosPolicy^ topic_data** `[get]`

Policy of the corresponding ***Topic*** *(p. 1258).*

6.204.1 Detailed Description

Entry created when a **Topic** (p. 1258) object discovered.

Data associated with the built-in topic **DDS::TopicBuiltinTopicDataTypeSupport::TOPIC_-TOPIC_NAME** (p. 228). It contains QoS policies and additional information that apply to the remote **DDS::Topic** (p. 1258).

Note: The DDS_TopicBuiltinTopicData built-in topic is meant to convey information about discovered Topics. This Topics 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. 1030) and **DDS::SubscriptionBuiltinTopicData** (p. 1233)). Therefore **TopicBuiltinTopicData** (p. 1268) DataReaders will not receive any data.

See also:

DDS::TopicBuiltinTopicDataTypeSupport::TOPIC_TOPIC_-NAME (p. 228)
DDS::TopicBuiltinTopicDataDataReader (p. 1273)

6.204.2 Member Data Documentation

6.204.2.1 BuiltinTopicKey_t DDS::TopicBuiltinTopicData::key

DCPS key to distinguish entries.

6.204.2.2 System::String ^ DDS::TopicBuiltinTopicData::name

Name of the **DDS::Topic** (p. 1258).

The length of this string is limited to 255 characters.

6.204.2.3 System::String ^ DDS::TopicBuiltinTopicData::type_name

Name of the type attached to the **DDS::Topic** (p. 1258).

The length of this string is limited to 255 characters.

6.204.2.4 DurabilityQosPolicy DDS::TopicBuiltinTopicData::durability

durability policy of the corresponding **Topic** (p. 1258)

6.204.2.5 DurabilityServiceQosPolicy DDS::TopicBuiltinTopicData::durability_service

durability service policy of the corresponding **Topic** (p. 1258)

6.204.2.6 DeadlineQosPolicy DDS::TopicBuiltinTopicData::deadline

Policy of the corresponding **Topic** (p. 1258).

6.204.2.7 LatencyBudgetQosPolicy DDS::TopicBuiltinTopicData::latency_budget

Policy of the corresponding **Topic** (p. 1258).

6.204.2.8 LivelinessQosPolicy DDS::TopicBuiltinTopicData::liveliness

Policy of the corresponding **Topic** (p. 1258).

6.204.2.9 ReliabilityQosPolicy DDS::TopicBuiltinTopicData::reliability

Policy of the corresponding **Topic** (p. 1258).

6.204.2.10 TransportPriorityQosPolicy DDS::TopicBuiltinTopicData::transport_priority

Policy of the corresponding **Topic** (p. 1258).

6.204.2.11 LifespanQosPolicy DDS::TopicBuiltinTopicData::lifespan

Policy of the corresponding **Topic** (p. 1258).

6.204.2.12 DestinationOrderQosPolicy DDS::TopicBuiltinTopicData::destination_order

Policy of the corresponding **Topic** (p. 1258).

6.204.2.13 HistoryQosPolicy DDS::TopicBuiltinTopicData::history

Policy of the corresponding **Topic** (p. 1258).

6.204.2.14 ResourceLimitsQosPolicy DDS::TopicBuiltinTopicData::resource_limits

Policy of the corresponding **Topic** (p. 1258).

6.204.2.15 OwnershipQosPolicy DDS::TopicBuiltinTopicData::ownership

Policy of the corresponding **Topic** (p. 1258).

6.204.3 Property Documentation

6.204.3.1 TopicDataQosPolicy^ DDS::TopicBuiltinTopicData::topic_data [get]

Policy of the corresponding **Topic** (p. 1258).

6.205 DDS::TopicBuiltinTopicDataDataReader Class Reference

Instantiates `DataReader` (p. 433) < **DDS::TopicBuiltinTopicData** (p. 1268) > .

`#include <managed_builtin.h>`

Inheritance diagram for DDS::TopicBuiltinTopicDataDataReader::

6.205.1 Detailed Description

Instantiates `DataReader` (p. 433) < **DDS::TopicBuiltinTopicData** (p. 1268) > .

DDS::DataReader (p. 433) of topic **DDS::TopicBuiltinTopicDataTypeSupport::TOPIC_TOPIC_NAME** (p. 228) used for accessing **DDS::TopicBuiltinTopicData** (p. 1268) of the remote **DDS::Topic** (p. 1258).

Note: The **DDS::TopicBuiltinTopicData** (p. 1268) built-in topic is meant to convey information about discovered Topics. This Topics 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. 1030) and **DDS::SubscriptionBuiltinTopicData** (p. 1233)). Therefore **TopicBuiltinTopicData** (p. 1268) DataReaders will not receive any data.

Instantiates:

<<*generic*>> *(p. 175)* **DDS::TypedDataReader** (p. 1338)

See also:

DDS::TopicBuiltinTopicData (p. 1268)
DDS::TopicBuiltinTopicDataTypeSupport::TOPIC_TOPIC_NAME (p. 228)

6.206 DDS::TopicBuiltinTopicDataSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < **DDS::TopicBuiltinTopicData** (p. 1268) > .

`#include <managed_builtin.h>`

6.206.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < **DDS::TopicBuiltinTopicData** (p. 1268) > .

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

DDS::TopicBuiltinTopicData (p. 1268)

6.207 DDS::TopicBuiltinTopicDataTypeSupport Class Reference

Instantiates `TypeSupport` (p. 1385) < **DDS::TopicBuiltinTopicData** (p. 1268) > .

`#include <managed_builtin.h>`

Inherits DDS::AbstractBuiltinTopicDataTypeSupport< T >.

Properties

^ static System::String^ **TOPIC_TOPIC_NAME** `[get]`

Topic *(p. 1258) topic name.*

6.207.1 Detailed Description

Instantiates `TypeSupport` (p. 1385) < **DDS::TopicBuiltinTopicData** (p. 1268) > .

Instantiates:

<<***generic***>> *(p. 175)* **FooTypeSupport** (p. 884)

See also:

DDS::TopicBuiltinTopicData (p. 1268)

6.208 DDS::TopicDataQosPolicy Class Reference

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 42) during discovery.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_topicdata_qos_policy_name** ()

  *Stringified human-readable name for **DDS::TopicDataQosPolicy*** (p. *1276*).

Public Attributes

- **ByteSeq^ 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. 42) during discovery.

Entity:

DDS::Topic (p. 1258)

Properties:

RxO (p. 268) = NO
Changeable (p. 269) = **YES** (p. 269)

See also:

DDS::DomainParticipant::get_builtin_subscriber (p. 632)

6.208.2 Usage

The purpose of this QoS is to allow the application to attach additional information to the created **DDS::Topic** (p. 1258) 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 Data Distribution Service.

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. 461), **DDS::DataWriterListener** (p. 524), or operations such as **DDS::DomainParticipant::ignore_topic** (p. 634), 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 Data Distribution Service stores the data placed in this policy in pre-allocated pools. It is therefore necessary to configure RTI Data Distribution Service 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. 701).

6.208.3 Member Data Documentation

6.208.3.1 ByteSeq ^ DDS::TopicDataQosPolicy::value

a sequence of octets

[**default**] empty (zero-length)

[**range**] Octet sequence of length [0,max_length]

6.209 DDS::TopicListener Class Reference

<<*interface*>> *(p. 175)* **DDS::Listener** (p. 952) for **DDS::Topic** (p. 1258) entities.

`#include <managed_topic.h>`

Inheritance diagram for DDS::TopicListener::

Public Member Functions

- ^ virtual void **on_inconsistent_topic** (**Topic**^ topic, **InconsistentTopicStatus**% status) override

  Handle the DDS::StatusKind::INCONSISTENT_TOPIC_STATUS status.

6.209.1 Detailed Description

<<*interface*>> *(p. 175)* **DDS::Listener** (p. 952) for **DDS::Topic** (p. 1258) entities.

Entity:

DDS::Topic (p. 1258)

Status:

DDS::StatusKind::INCONSISTENT_TOPIC_STATUS,
DDS::InconsistentTopicStatus (p. 903)

This is the interface that can be implemented by an application-provided class and then registered with the **DDS::Topic** (p. 1258) such that the application can be notified by RTI Data Distribution Service of relevant status changes.

See also:

Status Kinds (p. 238)
DDS::Listener (p. 952)
DDS::Topic::set_listener (p. 1263)
Operations Allowed in Listener Callbacks (p. 954)

6.209.2 Member Function Documentation

6.209.2.1 virtual void DDS::TopicListener::on_inconsistent_topic (Topic^ *topic*, InconsistentTopicStatus% *status*) [pure virtual]

Handle the DDS::StatusKind::INCONSISTENT_TOPIC_STATUS status.

This callback is called when a remote **DDS::Topic** (p. 1258) is discovered but is inconsistent with the locally created **DDS::Topic** (p. 1258) of the same topic name.

Parameters:

topic <<*out*>> *(p. 176)* Locally created **DDS::Topic** (p. 1258) that triggers the listener callback

status <<*out*>> *(p. 176)* Current inconsistent status of locally created **DDS::Topic** (p. 1258)

Implemented in **DDS::DomainParticipantListener** (p. 678).

6.210 DDS::TopicQos Class Reference

QoS policies supported by a **DDS::Topic** (p. 1258) entity.

`#include <managed_topic.h>`

Public Attributes

- ^ **TopicDataQosPolicy^ topic_data**
  Topic *(p. 1258) data policy,* ***TOPIC_DATA*** *(p. 274).*
- ^ **DurabilityQosPolicy durability**
  Durability policy, ***DURABILITY*** *(p. 276).*
- ^ **DurabilityServiceQosPolicy durability_service**
  DurabilityService policy, ***DURABILITY_SERVICE*** *(p. 297).*
- ^ **DeadlineQosPolicy deadline**
  Deadline policy, ***DEADLINE*** *(p. 281).*
- ^ **LatencyBudgetQosPolicy latency_budget**
  Latency budget policy, ***LATENCY_BUDGET*** *(p. 282).*
- ^ **LivelinessQosPolicy liveliness**
  Liveliness policy, ***LIVELINESS*** *(p. 286).*
- ^ **ReliabilityQosPolicy reliability**
  Reliability policy, ***RELIABILITY*** *(p. 290).*
- ^ **DestinationOrderQosPolicy destination_order**
  Destination order policy, ***DESTINATION_ORDER*** *(p. 292).*
- ^ **HistoryQosPolicy history**
  History policy, ***HISTORY*** *(p. 294).*
- ^ **ResourceLimitsQosPolicy resource_limits**
  Resource limits policy, ***RESOURCE_LIMITS*** *(p. 298).*
- ^ **TransportPriorityQosPolicy transport_priority**
  Transport priority policy, ***TRANSPORT_PRIORITY*** *(p. 300).*
- ^ **LifespanQosPolicy lifespan**
  Lifespan policy, ***LIFESPAN*** *(p. 301).*

^ **OwnershipQosPolicy ownership**

Ownership policy, ***OWNERSHIP*** *(p. 283).*

6.210.1 Detailed Description

QoS policies supported by a **DDS::Topic** (p. 1258) entity.

You must set certain members in a consistent manner:

length of **DDS::TopicQos::topic_data** (p. 1281) .value <= **DDS::DomainParticipantQos::resource_limits** (p. 686) .topic_data_max_length

If any of the above are not true, **DDS::Topic::set_qos** (p. 1261), **DDS::Topic::set_qos_with_profile** (p. 1262) and **DDS::DomainParticipant::set_default_topic_qos** (p. 608) will fail with **DDS::Retcode_InconsistentPolicy** (p. 1119) and **DDS::DomainParticipant::create_topic** (p. 621) will return NULL.

Entity:

DDS::Topic (p. 1258)

See also:

QoS Policies (p. 260) allowed ranges within each Qos.

6.210.2 Member Data Documentation

6.210.2.1 TopicDataQosPolicy ^ DDS::TopicQos::topic_data

Topic (p. 1258) data policy, **TOPIC_DATA** (p. 274).

6.210.2.2 DurabilityQosPolicy DDS::TopicQos::durability

Durability policy, **DURABILITY** (p. 276).

6.210.2.3 DurabilityServiceQosPolicy DDS::TopicQos::durability_service

DurabilityService policy, **DURABILITY_SERVICE** (p. 297).

6.210.2.4 DeadlineQosPolicy DDS::TopicQos::deadline

Deadline policy, **DEADLINE** (p. 281).

6.210.2.5 LatencyBudgetQosPolicy DDS::TopicQos::latency_budget

Latency budget policy, **LATENCY_BUDGET** (p. 282).

6.210.2.6 LivelinessQosPolicy DDS::TopicQos::liveliness

Liveliness policy, **LIVELINESS** (p. 286).

6.210.2.7 ReliabilityQosPolicy DDS::TopicQos::reliability

Reliability policy, **RELIABILITY** (p. 290).

6.210.2.8 DestinationOrderQosPolicy DDS::TopicQos::destination_order

Destination order policy, **DESTINATION_ORDER** (p. 292).

6.210.2.9 HistoryQosPolicy DDS::TopicQos::history

History policy, **HISTORY** (p. 294).

6.210.2.10 ResourceLimitsQosPolicy DDS::TopicQos::resource_limits

Resource limits policy, **RESOURCE_LIMITS** (p. 298).

6.210.2.11 TransportPriorityQosPolicy DDS::TopicQos::transport_priority

Transport priority policy, **TRANSPORT_PRIORITY** (p. 300).

6.210.2.12 LifespanQosPolicy DDS::TopicQos::lifespan

Lifespan policy, **LIFESPAN** (p. 301).

6.210.2.13 OwnershipQosPolicy DDS::TopicQos::ownership

Ownership policy, **OWNERSHIP** (p. 283).

6.211 DDS::TransportBuiltinKindAlias Class Reference

Bits in DDS::TransportBuiltinKindMask .

`#include <managed_infrastructure.h>`

Static Public Attributes

- ^ static System::String^ **TRANSPORTBUILTIN_SHMEM_ALIAS**
  Alias name for the shared memory built-in transport.
- ^ static System::String^ **TRANSPORTBUILTIN_UDPv4_ALIAS**
  Alias name for the UDPv4 built-in transport.
- ^ static System::String^ **TRANSPORTBUILTIN_UDPv6_ALIAS**
  Alias name for the UDPv6 built-in transport.

6.211.1 Detailed Description

Bits in DDS::TransportBuiltinKindMask .

QoS:

DDS::TransportBuiltinQosPolicy (p. 1285)

6.212 DDS::TransportBuiltinQosPolicy Struct Reference

Specifies which built-in transports are used.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_transportbuiltin_qos_policy_name** ()

  *Stringified human-readable name for **DDS::TransportBuiltinQosPolicy*** *(p. 1285)**.*

Public Attributes

- System::Int32 **mask**

  *Specifies the built-in transports that are registered automatically when the **DDS::DomainParticipant*** *(p. 577)* *is enabled.*

6.212.1 Detailed Description

Specifies which built-in transports are used.

Three different transport plug-ins are built into the core RTI Data Distribution Service 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. 577). 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. 577)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)

6.212.2 Member Data Documentation

6.212.2.1 System::Int32 DDS::TransportBuiltinQosPolicy::mask

Specifies the built-in transports that are registered automatically when the **DDS::DomainParticipant** (p. 577) is enabled.

RTI Data Distribution Service provides several built-in transports. Only those that are specified with this mask are registered automatically when the **DDS::DomainParticipant** (p. 577) is enabled.

[default] DDS::TransportBuiltinKindMask::TRANSPORTBUILTIN_MASK_DEFAULT

6.213 DDS::TransportMulticastQosPolicy Class Reference

Specifies the multicast address on which a **DDS::DataReader** (p. 433) 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. 577) level) transports with which to receive the multicast data.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_transportmulticast_qos_policy_name** ()

  *Stringified human-readable name for **DDS::TransportMulticastQosPolicy** (p. 1287).*

Public Attributes

- **TransportMulticastSettingsSeq^ value**

  A sequence of multicast communications settings.

6.213.1 Detailed Description

Specifies the multicast address on which a **DDS::DataReader** (p. 433) 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. 577) level) transports with which to receive the multicast data.

By default, a **DDS::DataWriter** (p. 499) will send individually addressed packets for each **DDS::DataReader** (p. 433) that subscribes to the topic of the **DataWriter** (p. 499) 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** (p. 499) 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. 499) 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. 1258).

Entity:

DDS::DataReader (p. 433)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)

6.213.2 Member Data Documentation

6.213.2.1 TransportMulticastSettingsSeq ^ DDS::TransportMulticastQosPolicy::value

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 four.

[default] Empty sequence.

6.214 DDS::TransportMulticastSettings_t Class Reference

Type representing a list of multicast locators.

`#include <managed_infrastructure.h>`

Public Attributes

- ^ **StringSeq^ transports**

  A sequence of transport aliases that specifies the transports on which to receive multicast *traffic for the entity.*

- ^ System::String^ **receive_address**

  The multicast group address on which the entity can receive data.

- ^ System::Int32 **receive_port**

  The multicast port on which the entity can receive data.

6.214.1 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. 1287)

6.214.2 Member Data Documentation

6.214.2.1 StringSeq ^ DDS::TransportMulticastSettings_t::transports

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.

Alias names for the builtin transports are defined in **TRANSPORT_BUILTIN** (p. 321).

[**default**] Empty sequence; i.e. all the transports available to the entity.

[**range**] Any sequence of non-null, non-empty strings.

6.214.2.2 System::String ^ DDS::TransportMulticastSettings_t::receive_address

The multicast group address on which the entity can receive data.

Must must be an address in the proper format (see **Address Format** (p. 314)).

[**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. 314)

6.214.2.3 System::Int32 DDS::TransportMulticastSettings_t::receive_port

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. 1427)).

[**range**] [0,0xffffffff]

6.215 DDS::TransportMulticastSettingsSeq Class Reference

Declares IDL `sequence<` **DDS::TransportMulticastSettings_t** (p. 1289) `>`.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::TransportMulticastSettingsSeq::

6.215.1 Detailed Description

Declares IDL `sequence<` **DDS::TransportMulticastSettings_t** (p. 1289) `>`.

Instantiates:

<<*generic*>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

DDS::TransportMulticastSettings_t (p. 1289)

6.216 DDS::TransportPriorityQosPolicy Struct Reference

This QoS policy allows the application to take advantage of transports that are capable of sending messages with different priorities.

`#include <managed_infrastructure.h>`

Static Public Member Functions

^ static System::String^ **get_transportpriority_qos_policy_name** ()

*Stringified human-readable name for **DDS::TransportPriorityQosPolicy** (p. 1292).*

Public Attributes

^ System::Int32 **value**

This policy is a hint to the infrastructure as to how to set the priority of the underlying transport used to send the data.

6.216.1 Detailed Description

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-**DDS::DataWriter** (p. 499) basis that the data sent by that **DDS::DataWriter** (p. 499) 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 **DDS::TypedDataWriter::write** (p. 1376). Also, many physical network transports themselves do not have a end-user controllable level of data packet priority.

Entity:

DDS::DataWriter (p. 499), **DDS::Topic** (p. 1258)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **YES** (p. 269)

6.216.2 Usage

In RTI Data Distribution Service, for the **DDS::UDPv4Transport** (p. 1388), the value set in the Transport Priority QoS policy is used in a `setsockopt` call to set the TOS (type of service) bits of the IPv4 header for datagrams sent by a **DDS::DataWriter** (p. 499). It is platform-dependent how and whether the `setsockopt` has an effect. On some platforms, such as Windows and Linux, external permissions must be given to the user application in order to set the TOS bits.

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. 499) to a **DDS::DataReader** (p. 433). All network elements, including switches and routers must have the capability and be enabled to actually use the TOS bits 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.

6.216.3 Member Data Documentation

6.216.3.1 System::Int32 DDS::TransportPriorityQosPolicy::value

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

6.217 DDS::TransportSelectionQosPolicy Class Reference

Specifies the physical transports a **DDS::DataWriter** (p. 499) or **DDS::DataReader** (p. 433) may use to send or receive data.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_transportselection_qos_policy_name** ()

  *Stringified human-readable name for **DDS::TransportSelectionQosPolicy** (p. 1294).*

Public Attributes

- **StringSeq^ enabled_transports**

  A sequence of transport aliases that specifies the transport instances available for use by the entity.

6.217.1 Detailed Description

Specifies the physical transports a **DDS::DataWriter** (p. 499) or **DDS::DataReader** (p. 433) 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 Data Distribution Service will use up to 4 transports to deliver data from a **DataWriter** (p. 499) to a **DataReader** (p. 433).

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. 499) to send data or by a **DDS::DataReader** (p. 433) to receive data.

Entity:

DDS::DataReader (p. 433), **DDS::DataWriter** (p. 499)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)

6.217.2 Member Data Documentation

6.217.2.1 StringSeq ^ DDS::TransportSelectionQosPolicy::enabled_transports

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. 577), 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. 577).

An empty sequence is a special value that specifies all the transports installed with the **DDS::DomainParticipant** (p. 577).

Alias names for the builtin transports are defined in **TRANSPORT_BUILTIN** (p. 321).

[**default**] Empty sequence; i.e. all the transports installed with and available to the **DDS::DomainParticipant** (p. 577).

[**range**] A sequence of non-null, non-empty strings.

See also:

DDS::DomainParticipantQos::transport_builtin (p. 685).

6.218 DDS::TransportUnicastQosPolicy Class Reference

Specifies a subset of transports and a port number that can be used by an **Entity** (p. 845) to receive data.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_transportunicast_qos_policy_name** ()

  *Stringified human-readable name for **DDS::TransportUnicastQosPolicy** (p. 1296).*

Public Attributes

- **TransportUnicastSettingsSeq^ value**

  A sequence of unicast communication settings.

6.218.1 Detailed Description

Specifies a subset of transports and a port number that can be used by an **Entity** (p. 845) to receive data.

Entity:

DDS::DomainParticipant (p. 577), **DDS::DataReader** (p. 433), **DDS::DataWriter** (p. 499)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)

6.218.2 Usage

RTI Data Distribution Service 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. 845) announces to remote applications a list of (up to 4) 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).

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** (p. 845) (see **DDS::TransportBuiltinQosPolicy** (p. 1285) and **DDS::TransportSelectionQosPolicy** (p. 1294)). Also, the associated ports are automatically determined (see **DDS::RtpsWellKnownPorts_t** (p. 1142)).

Use this QoS policy to manually set the receive address list for an **Entity** (p. 845). You may optionally set a port to use a non-default receive port as well. Only the first 4 addresses will be used.

RTI Data Distribution Service will create a receive thread for every unique port number that it encounters (on a per transport basis).

- For a **DDS::DomainParticipant** (p. 577), 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. 433), 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. 577).

- For a reliable **DDS::DataWriter** (p. 499), 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. 577).

6.218.3 Member Data Documentation

6.218.3.1 TransportUnicastSettingsSeq ^ DDS::TransportUnicastQosPolicy::value

A sequence of unicast communication settings.

An empty sequence means that applicable defaults specified by elsewhere (e.g. **DDS::DomainParticipantQos::default_unicast** (p. 685)) should be used.

The RTPS wire protocol currently limits the maximum number of unicast locators to four.

[**default**] Empty sequence.

See also:

DDS::DomainParticipantQos::default_unicast (p. 685)

6.219 DDS::TransportUnicastSettings_t Class Reference

Type representing a list of unicast locators.

`#include <managed_infrastructure.h>`

Public Attributes

- ^ **StringSeq^ transports**

  A sequence of transport aliases that specifies the unicast interfaces on which to receive unicast *traffic for the entity.*

- ^ System::Int32 **receive_port**

  The unicast port on which the entity can receive data.

6.219.1 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. 1296)

6.219.2 Member Data Documentation

6.219.2.1 StringSeq ^ DDS::TransportUnicastSettings_t::transports

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.

Alias names for the builtin transports are defined in **TRANSPORT_BUILTIN** (p. 321).

[**default**] Empty sequence; i.e. all the transports available to the entity.

[**range**] Any sequence of non-null, non-empty strings.

6.219.2.2 System::Int32 DDS::TransportUnicastSettings_t::receive_port

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. 1427).

[**range**] [0,0xffffffff]

See also:

DDS::WireProtocolQosPolicy::participant_id (p. 1427).

6.220 DDS::TransportUnicastSettingsSeq Class Reference

Declares IDL `sequence<` **DDS::TransportUnicastSettings_t** (p. 1298) `>`.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::TransportUnicastSettingsSeq::

6.220.1 Detailed Description

Declares IDL `sequence<` **DDS::TransportUnicastSettings_t** (p. 1298) `>`.

Instantiates:

<<generic>> (p. *175*) **DDS::Sequence** (p. 1163)

See also:

DDS::TransportUnicastSettings_t (p. 1298)

6.221 DDS::TypeCode Class 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** (p. 196) or to modify types you define yourself at runtime.

`#include <managed_typecode.h>`

Public Member Functions

- ^ **TCKind kind** ()

  Gets the DDS::TCKind value of a type code.

- ^ System::Boolean **equal** (**TypeCode**^ tc)

  *Compares two **DDS::TypeCode** (p. 1301) objects for equality.*

- ^ System::String^ **name** ()

  *Retrieves the simple name identifying this **DDS::TypeCode** (p. 1301) object within its enclosing scope.*

- ^ System::UInt32 **member_count** ()

  Returns the number of members of the type code.

- ^ System::String^ **member_name** (System::UInt32 index)

  Returns the name of a type code member identified by the given index.

- ^ System::UInt32 **find_member_by_name** (System::String^ name)

  Get the index of the member of the given name.

- ^ **DDS::TypeCode**^ **member_type** (System::UInt32 index)

  *Retrieves the **DDS::TypeCode** (p. 1301) object describing the type of the member identified by the given index.*

- ^ System::UInt32 **member_label_count** (System::UInt32 index)

  Returns the number of labels associated to the index-th union member.

- ^ System::Int32 **member_label** (System::UInt32 member_index, System::UInt32 label_index)

  Return the label_index-th label associated to the member_index-th member.

- ^ System::Int32 **member_ordinal** (System::UInt32 index)

  Returns the ordinal that corresponds to the index-th enum value.

^ System::Boolean **is_member_key** (System::UInt32 index)

Function that tells if a member is a key or not.

^ System::Boolean **is_member_required** (System::UInt32 index)

Indicates whether a given member of a type is required to be present in every sample of that type.

^ System::Boolean **is_member_pointer** (System::UInt32 index)

Function that tells if a member is a pointer or not.

^ System::Boolean **is_member_bitfield** (System::UInt32 index)

Function that tells if a member is a bitfield or not.

^ System::Int16 **member_bitfield_bits** (System::UInt32 index)

Returns the number of bits of a bitfield member.

^ **Visibility member_visibility** (System::UInt32 index)

Returns the constant that indicates the visibility of the index-th member.

^ **TypeCode^ discriminator_type** ()

Returns the discriminator type code.

^ System::UInt32 **length** ()

Returns the number of elements in the type described by this type code.

^ System::UInt32 **array_dimension_count** ()

This function returns the number of dimensions of an array type code.

^ System::UInt32 **array_dimension** (System::UInt32 index)

This function returns the index-th dimension of an array type code.

^ System::UInt32 **element_count** ()

The number of elements in an array.

^ **DDS::TypeCode^ content_type** ()

*Returns the **DDS::TypeCode** (p. 1301) object representing the type for the members of the object described by this **DDS::TypeCode** (p. 1301) object.*

^ System::Boolean **is_alias_pointer** ()

Function that tells if an alias is a pointer or not.

^ System::Int32 **default_index** ()

Returns the index of the default member, or -1 if there is no default member.

^ **DDS::TypeCode^ concrete_base_type** ()

*Returns the **DDS::TypeCode** (p. 1301) that describes the concrete base type of the value type that this **DDS::TypeCode** (p. 1301) object describes.*

^ **ValueModifier type_modifier** ()

*Returns a constant indicating the modifier of the value type that this **DDS::TypeCode** (p. 1301) object describes.*

^ System::Int32 **member_id** (System::UInt32 index)

Returns the ID of a sparse type code member identified by the given index.

^ System::UInt32 **find_member_by_id** (System::Int32 id)

Get the index of the member of the given ID.

^ System::UInt32 **add_member_to_enum** (System::String^ name, System::Int32 ordinal)

*Add a new enumerated constant to this enum **DDS::TypeCode** (p. 1301).*

^ System::UInt32 **add_member** (System::String^ name, System::Int32 id, **DDS::TypeCode**^ tc, System::Byte member_flags)

*Add a new member to this **DDS::TypeCode** (p. 1301).*

^ System::UInt32 **add_member_ex** (System::String^ name, System::Int32 id, **DDS::TypeCode**^ tc, System::Byte member_flags, **Visibility** visibility, System::Boolean is_pointer, System::Int16 bits)

*Add a new member to this **DDS::TypeCode** (p. 1301).*

^ void **print_IDL** (System::UInt32 indent)

*Prints a **DDS::TypeCode** (p. 1301) in a pseudo-IDL notation.*

Public Attributes

^ **TypeCode^ TC_NULL**

Basic null type.

^ **TypeCode^ TC_NULL**

Basic 16-bit signed integer type.

^ **TypeCode^ TC_LONG**

Basic 32-bit signed integer type.

^ **TypeCode^ TC_USHORT**

Basic unsigned 16-bit integer type.

^ **TypeCode^ TC_ULONG**

Basic unsigned 32-bit integer type.

^ **TypeCode^ TC_FLOAT**

Basic 32-bit floating point type.

^ **TypeCode^ TC_DOUBLE**

Basic 64-bit floating point type.

^ **TypeCode^ TC_BOOLEAN**

Basic Boolean type.

^ **TypeCode^ TC_CHAR**

Basic single-byte character type.

^ **TypeCode^ TC_OCTET**

Basic octet/byte type.

^ **TypeCode^ TC_LONGLONG**

Basic 64-bit integer type.

^ **TypeCode^ TC_ULONGLONG**

Basic unsigned 64-bit integer type.

^ **TypeCode^ TC_LONGDOUBLE**

Basic 128-bit floating point type.

^ **TypeCode^ TC_WCHAR**

Basic four-byte character type.

^ System::Int32 **MEMBER_ID_INVALID**

*A sentinel indicating an invalid **DDS::TypeCode** (p. 1301) member ID.*

^ System::UInt32 **INDEX_INVALID**

*A sentinel indicating an invalid **DDS::TypeCode** (p. 1301) member index.*

^ System::Int16 **NOT_BITFIELD**

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

ˆ System::Byte **NONKEY_MEMBER**

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

ˆ System::Byte **KEY_MEMBER**

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

ˆ System::Byte **NONKEY_REQUIRED_MEMBER**

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

6.221.1 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** (p. 196) or to modify types you define yourself at runtime.

You create **DDS::TypeCode** (p. 1301) objects using the **DDS::TypeCodeFactory** (p. 1327) singleton. Then you can use the methods on *this* class to inspect and modify the data type definition.

This class is based on a similar class from CORBA.

MT Safety:

SAFE for read-only access, UNSAFE for modification. Modifying a single **DDS::TypeCode** (p. 1301) object concurrently from multiple threads is *unsafe*. Modifying a **DDS::TypeCode** (p. 1301) from a single thread while concurrently reading the state of that **DDS::TypeCode** (p. 1301) from another thread is also *unsafe*. However, reading the state of a **DDS::TypeCode** (p. 1301) concurrently from multiple threads, without any modification, is *safe*.

Examples:

HelloWorld.cpp.

6.221.2 Member Function Documentation

6.221.2.1 TCKind DDS::TypeCode::kind ()

Gets the DDS::TCKind value of a type code.

Retrieves the kind of this **DDS::TypeCode** (p. 1301) object. The kind of a type code determines which **DDS::TypeCode** (p. 1301) methods may legally be invoked on it.

MT Safety:

SAFE.

Returns:

The type code kind.

6.221.2.2 System::Boolean DDS::TypeCode::equal (TypeCode^ *tc*)

Compares two **DDS::TypeCode** (p. 1301) objects for equality.

MT Safety:

SAFE.

Parameters:

tc <<*in*>> *(p. 175)* Type code that will be compared with this **DDS::TypeCode** (p. 1301).

Exceptions:

DDS::ExceptionCode_t::BAD_PARAM_SYSTEM_EXCEPTION_CODE if tc is null.

Returns:

true if the type codes are equal. Otherwise, false.

6.221.2.3 System::String ^ DDS::TypeCode::name ()

Retrieves the simple name identifying this **DDS::TypeCode** (p. 1301) object within its enclosing scope.

Precondition:

self kind is DDS::TCKind::TK_STRUCT, DDS::TCKind::TK_UNION, DDS::TCKind::TK_ENUM, DDS::TCKind::TK_VALUE, DDS::TCKind::TK_SPARSE or DDS::TCKind::TK_ALIAS.

MT Safety:

SAFE.

Exceptions:

DDS::ExceptionCode_t::BADKIND_USER_EXCEPTION_CODE if the method is invoked on an inappropriate kind of **TypeCode** (p. 1301) object.

Returns:

Name of the type code if no errors.

6.221.2.4 System::UInt32 DDS::TypeCode::member_count ()

Returns the number of members of the type code.

The method member_count can be invoked on structure, union, and enumeration **DDS::TypeCode** (p. 1301) objects.

Precondition:

self kind is DDS::TCKind::TK_STRUCT, DDS::TCKind::TK_UNION, DDS::TCKind::TK_ENUM, DDS::TCKind::TK_VALUE or DDS::TCKind::TK_SPARSE.

MT Safety:

SAFE.

Exceptions:

DDS::ExceptionCode_t::BADKIND_USER_EXCEPTION_CODE if the method is invoked on an inappropriate kind of **TypeCode** (p. 1301) object.

Returns:

The number of members constituting the type described by this **DDS::TypeCode** (p. 1301) object if no errors.

6.221.2.5 System::String ^ DDS::TypeCode::member_name (System::UInt32 *index*)

Returns the name of a type code member identified by the given index.

The method member_name can be invoked on structure, union, and enumeration **DDS::TypeCode** (p. 1301) objects.

Precondition:

self kind is DDS::TCKind::TK_STRUCT, DDS::TCKind::TK_UNION, DDS::TCKind::TK_ENUM, DDS::TCKind::TK_VALUE or DDS::TCKind::TK_SPARSE.
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters:

index *<<in>>* *(*p. 175*)* Member index in the interval [0,(member count-1)].

Exceptions:

DDS::ExceptionCode_t::BADKIND_USER_EXCEPTION_CODE if the method is invoked on an inappropriate kind of **TypeCode** (p. 1301) object.

DDS::ExceptionCode_t::BOUNDS_USER_EXCEPTION_CODE - if the index parameter/s are out of range.

Returns:

Name of the member if no errors.

6.221.2.6 System::UInt32 DDS::TypeCode::find_member_by_name (System::String^ *name*)

Get the index of the member of the given name.

MT Safety:

SAFE.

6.221.2.7 DDS::TypeCode ^ DDS::TypeCode::member_type (System::UInt32 *index*)

Retrieves the **DDS::TypeCode** (p. 1301) object describing the type of the member identified by the given index.

The method member_type can be invoked on structure and union type codes.

Precondition:

self kind is DDS::TCKind::TK_STRUCT, DDS::TCKind::TK_UNION, DDS::TCKind::TK_VALUE or DDS::TCKind::TK_SPARSE.
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters:

index *<<in>>* *(p. 175)* Member index in the interval [0,(member count-1)].

Exceptions:

DDS::ExceptionCode_t::BADKIND_USER_EXCEPTION_CODE if the method is invoked on an inappropriate kind of **TypeCode** (p. 1301) object.

DDS::ExceptionCode_t::BOUNDS_USER_EXCEPTION_CODE - if the index parameter/s are out of range.

Returns:

The **DDS::TypeCode** (p. 1301) object describing the member at the given index if no errors.

6.221.2.8 System::UInt32 DDS::TypeCode::member_label_count (System::UInt32 *index*)

Returns the number of labels associated to the index-th union member.

The method can be invoked on union **DDS::TypeCode** (p. 1301) objects.

This function is an RTI Data Distribution Service extension to the CORBA Type Code Specification.

Precondition:

self kind is DDS::TCKind::TK_UNION.
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters:

index *<<**in**>>* *(p. 175)* Member index in the interval [0,(member count-1)].

Exceptions:

DDS::ExceptionCode_t::BADKIND_USER_EXCEPTION_CODE if the method is invoked on an inappropriate kind of **TypeCode** (p. 1301) object.

DDS::ExceptionCode_t::BOUNDS_USER_EXCEPTION_CODE - if the index parameter/s are out of range.

Returns:

Number of labels if no errors.

6.221.2.9 System::Int32 DDS::TypeCode::member_label (System::UInt32 *member_index*, System::UInt32 *label_index*)

Return the label_index-th label associated to the member_index-th member.

This method has been modified for RTI Data Distribution Service from the CORBA Type code Specification.

Example:

case 1: Label index 0

case 2: Label index 1

short short_member;

The method can be invoked on union **DDS::TypeCode** (p. 1301) objects.

Precondition:

self kind is DDS::TCKind::TK_UNION.
The member_index param must be in the interval [0,(member count-1)].
The label_index param must be in the interval [0,(member labels count-1)].

MT Safety:

SAFE.

Parameters:

member_index *<<**in**>>* *(p. 175)* Member index.

label_index *<<**in**>>* *(p. 175)* Label index.

Exceptions:

DDS::ExceptionCode_t::BADKIND_USER_EXCEPTION_CODE if the method is invoked on an inappropriate kind of **TypeCode** (p. 1301) object.

DDS::ExceptionCode_t::BOUNDS_USER_EXCEPTION_CODE - if the index parameter/s are out of range.

Returns:

The evaluated value of the label if no errors.

6.221.2.10 System::Int32 DDS::TypeCode::member_ordinal (System::UInt32 *index*)

Returns the ordinal that corresponds to the index-th enum value.

The method can be invoked on enum **DDS::TypeCode** (p. 1301) objects.

This function is an RTI Data Distribution Service extension to the CORBA Type Code Specification.

Precondition:

self kind is DDS::TCKind::TK_ENUM.
Member index in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters:

index *<<in>>* *(p. 175)* Member index in the interval [0,(member count-1)].

Exceptions:

DDS::ExceptionCode_t::BADKIND_USER_EXCEPTION_CODE if the method is invoked on an inappropriate kind of **TypeCode** (p. 1301) object.

DDS::ExceptionCode_t::BOUNDS_USER_EXCEPTION_CODE - if the index parameter/s are out of range.

Returns:

Ordinal that corresponds to the index-th enumerator if no errors.

6.221.2.11 System::Boolean DDS::TypeCode::is_member_key (System::UInt32 *index*)

Function that tells if a member is a key or not.

This function is an RTI Data Distribution Service extension to the CORBA Type Code Specification.

Precondition:

self kind is DDS::TCKind::TK_STRUCT, DDS::TCKind::TK_VALUE or DDS::TCKind::TK_SPARSE.
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters:

index *<<in>>* *(*p. 175*)* Member index in the interval [0,(member count-1)].

Exceptions:

DDS::ExceptionCode_t::BADKIND_USER_EXCEPTION_CODE if the method is invoked on an inappropriate kind of **TypeCode** (p. 1301) object.

DDS::ExceptionCode_t::BOUNDS_USER_EXCEPTION_CODE - if the index parameter/s are out of range.

Returns:

true if the member is a key. Otherwise, false.

6.221.2.12 System::Boolean DDS::TypeCode::is_member_required (System::UInt32 *index*)

Indicates whether a given member of a type is required to be present in every sample of that type.

Which fields are required depends on the DDS::TCKind of the type. For example, in a type of kind DDS::TCKind::TK_SPARSE, key fields are required. In DDS::TCKind::TK_STRUCT and DDS::TCKind::TK_VALUE types, all fields are required.

MT Safety:

SAFE.

6.221.2.13 System::Boolean DDS::TypeCode::is_member_pointer (System::UInt32 *index*)

Function that tells if a member is a pointer or not.

The method is_member_pointer can be invoked on union and structs type objects

This function is an RTI Data Distribution Service extension to the CORBA Type Code Specification.

Precondition:

self kind is DDS::TCKind::TK_STRUCT, DDS::TCKind::TK_UNION or DDS::TCKind::TK_VALUE.
The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters:

index *<<in>>* *(p. 175)* Index of the member for which type information is begin requested.

Exceptions:

DDS::ExceptionCode_t::BADKIND_USER_EXCEPTION_CODE if the method is invoked on an inappropriate kind of **TypeCode** (p. 1301) object.

DDS::ExceptionCode_t::BOUNDS_USER_EXCEPTION_CODE - if the index parameter/s are out of range.

Returns:

true if the member is a pointer. Otherwise, false.

6.221.2.14 System::Boolean DDS::TypeCode::is_member_bitfield (System::UInt32 *index*)

Function that tells if a member is a bitfield or not.

The method can be invoked on struct type objects.

This function is an RTI Data Distribution Service extension to the CORBA Type Code Specification.

Precondition:

self kind is DDS::TCKind::TK_STRUCT or DDS::TCKind::TK_VALUE. The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters:

index *<<in>>* *(p. 175)* Member index in the interval [0,(member count-1)].

Exceptions:

DDS::ExceptionCode_t::BADKIND_USER_EXCEPTION_CODE if the method is invoked on an inappropriate kind of **TypeCode** (p. 1301) object.

DDS::ExceptionCode_t::BOUNDS_USER_EXCEPTION_CODE - if the index parameter/s are out of range.

Returns:

true if the member is a bitfield. Otherwise, false.

6.221.2.15 System::Int16 DDS::TypeCode::member_bitfield_bits (System::UInt32 *index*)

Returns the number of bits of a bitfield member.

The method can be invoked on struct type objects.

This function is an RTI Data Distribution Service extension to the CORBA Type Code Specification.

Precondition:

self kind is DDS::TCKind::TK_STRUCT or DDS::TCKind::TK_VALUE. The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters:

index *<<in>>* *(p. 175)* Member index in the interval [0,(member count-1)].

Exceptions:

DDS::ExceptionCode_t::BADKIND_USER_EXCEPTION_CODE if the method is invoked on an inappropriate kind of **TypeCode** (p. 1301) object.

DDS::ExceptionCode_t::BOUNDS_USER_EXCEPTION_CODE - if the index parameter/s are out of range.

Returns:

The number of bits of the bitfield or **DDS::TypeCode::NOT_BITFIELD** (p. 65) if the member is not a bitfield.

6.221.2.16 Visibility DDS::TypeCode::member_visibility (System::UInt32 *index*)

Returns the constant that indicates the visibility of the index-th member.

Precondition:

self kind is DDS::TCKind::TK_VALUE. The index param must be in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters:

index <<*in*>> *(p. 175)* Member index in the interval [0,(member count-1)].

Exceptions:

DDS::ExceptionCode_t::BADKIND_USER_EXCEPTION_CODE if the method is invoked on an inappropriate kind of **TypeCode** (p. 1301) object.

DDS::ExceptionCode_t::BOUNDS_USER_EXCEPTION_CODE - if the index parameter/s are out of range.

Returns:

One of the following constants: DDS::Visibility::PRIVATE_MEMBER or DDS::Visibility::PUBLIC_MEMBER.

6.221.2.17 TypeCode ^ DDS::TypeCode::discriminator_type ()

Returns the discriminator type code.

The method discriminator_type can be invoked only on union **DDS::TypeCode** (p. 1301) objects.

Precondition:

self kind is DDS::TCKind::TK_UNION.

MT Safety:

SAFE.

Exceptions:

DDS::ExceptionCode_t::BADKIND_USER_EXCEPTION_CODE if the method is invoked on an inappropriate kind of **TypeCode** (p. 1301) object.

Returns:

DDS::TypeCode (p. 1301) object describing the discriminator of the union type if no errors.

6.221.2.18 System::UInt32 DDS::TypeCode::length ()

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::TCKind::TK_ARRAY, DDS::TCKind::TK_SEQUENCE, DDS::TCKind::TK_STRING or DDS::TCKind::TK_WSTRING.

MT Safety:

SAFE.

Exceptions:

DDS::ExceptionCode_t::BADKIND_USER_EXCEPTION_CODE if the method is invoked on an inappropriate kind of **TypeCode** (p. 1301) object.

Returns:

The bound for strings and sequences, or the number of elements for arrays if no errors.

6.221.2.19 System::UInt32 DDS::TypeCode::array_dimension_count ()

This function returns the number of dimensions of an array type code.

This function is an RTI Data Distribution Service extension to the CORBA Type Code Specification.

Precondition:

self kind is DDS::TCKind::TK_ARRAY.

MT Safety:

SAFE.

Exceptions:

DDS::ExceptionCode_t::BADKIND_USER_EXCEPTION_CODE if the method is invoked on an inappropriate kind of **TypeCode** (p. 1301) object.

Returns:

Number of dimensions if no errors.

6.221.2.20 System::UInt32 DDS::TypeCode::array_dimension (System::UInt32 *index*)

This function returns the index-th dimension of an array type code.

This function is an RTI Data Distribution Service extension to the CORBA Type Code Specification.

Precondition:

self kind is DDS::TCKind::TK_ARRAY.
Dimension index in the interval [0,(dimensions count-1)].

MT Safety:

SAFE.

Parameters:

index *<<in>>* *(p. 175)* Dimension index in the interval [0,(dimensions count-1)].

Exceptions:

DDS::ExceptionCode_t::BADKIND_USER_EXCEPTION_CODE if the method is invoked on an inappropriate kind of **TypeCode** (p. 1301) object.

DDS::ExceptionCode_t::BOUNDS_USER_EXCEPTION_CODE - if the index parameter/s are out of range.

Returns:

Requested dimension if no errors.

6.221.2.21 System::UInt32 DDS::TypeCode::element_count ()

The number of elements in an array.

This operation isn't relevant for other kinds of types.

MT Safety:

SAFE.

6.221.2.22 DDS::TypeCode ^ DDS::TypeCode::content_type ()

Returns the **DDS::TypeCode** (p. 1301) object representing the type for the members of the object described by this **DDS::TypeCode** (p. 1301) object.

For sequences and arrays, it returns the element type. For aliases, it returns the original type.

Precondition:

self kind is DDS::TCKind::TK_ARRAY, DDS::TCKind::TK_SEQUENCE or DDS::TCKind::TK_ALIAS.

MT Safety:

SAFE.

Exceptions:

DDS::ExceptionCode_t::BADKIND_USER_EXCEPTION_CODE if the method is invoked on an inappropriate kind of **TypeCode** (p. 1301) object.

Returns:

A **DDS::TypeCode** (p. 1301) object representing the element type for sequences and arrays, and the original type for aliases.

6.221.2.23 System::Boolean DDS::TypeCode::is_alias_pointer ()

Function that tells if an alias is a pointer or not.

This function is an RTI Data Distribution Service extension to the CORBA Type Code Specification.

Precondition:

self kind is DDS::TCKind::TK_ALIAS.

MT Safety:

SAFE.

Exceptions:

DDS::ExceptionCode_t::BADKIND_USER_EXCEPTION_CODE if the method is invoked on an inappropriate kind of **TypeCode** (p. 1301) object.

Returns:

true if an alias is a pointer to the aliased type. Otherwise, false.

6.221.2.24 System::Int32 DDS::TypeCode::default_index ()

Returns the index of the default member, or -1 if there is no default member.

The method default_index can be invoked only on union **DDS::TypeCode** (p. 1301) objects.

Precondition:

self kind is DDS::TCKind::TK_UNION

MT Safety:

SAFE.

Exceptions:

DDS::ExceptionCode_t::BADKIND_USER_EXCEPTION_CODE if the method is invoked on an inappropriate kind of **TypeCode** (p. 1301) object.

Returns:

The index of the default member, or -1 if there is no default member.

6.221.2.25 DDS::TypeCode ^ DDS::TypeCode::concrete_base_type ()

Returns the **DDS::TypeCode** (p. 1301) that describes the concrete base type of the value type that this **DDS::TypeCode** (p. 1301) object describes.

Precondition:

self kind is DDS::TCKind::TK_VALUE or DDS::TCKind::TK_SPARSE.

MT Safety:

SAFE.

Exceptions:

DDS::ExceptionCode_t::BADKIND_USER_EXCEPTION_CODE if the method is invoked on an inappropriate kind of **TypeCode** (p. 1301) object.

Returns:

DDS::TypeCode (p. 1301) that describes the concrete base type or null if there is no a concrete base type.

6.221.2.26 ValueModifier DDS::TypeCode::type_modifier ()

Returns a constant indicating the modifier of the value type that this **DDS::TypeCode** (p. 1301) object describes.

Precondition:

self kind is DDS::TCKind::TK_VALUE.

MT Safety:

SAFE.

Exceptions:

DDS::ExceptionCode_t::BADKIND_USER_EXCEPTION_CODE if the method is invoked on an inappropriate kind of **TypeCode** (p. 1301) object.

Returns:

One of the following type modifiers: DDS::ValueModifier::VM_NONE, DDS::ValueModifier::VM_ABSTRACT, DDS::ValueModifier::VM_CUSTOM or DDS::ValueModifier::VM_TRUNCATABLE.

6.221.2.27 System::Int32 DDS::TypeCode::member_id (System::UInt32 *index*)

Returns the ID of a sparse type code member identified by the given index.

The method can be invoked on sparse **DDS::TypeCode** (p. 1301) objects.

This function is an RTI Data Distribution Service extension to the CORBA Type Code Specification.

Precondition:

self kind is DDS::TCKind::TK_SPARSE.
Member index in the interval [0,(member count-1)].

MT Safety:

SAFE.

Parameters:

index *<<in>> (p. 175)* Member index in the interval [0,(member count-1)].

Exceptions:

DDS::ExceptionCode_t::BADKIND_USER_EXCEPTION_CODE if the method is invoked on an inappropriate kind of **TypeCode** (p. 1301) object.

DDS::ExceptionCode_t::BOUNDS_USER_EXCEPTION_CODE - if the index parameter/s are out of range.

Returns:

ID of the member if no errors.

6.221.2.28 System::UInt32 DDS::TypeCode::find_member_by_id (System::Int32 *id*)

Get the index of the member of the given ID.

MT Safety:

SAFE.

6.221.2.29 System::UInt32 DDS::TypeCode::add_member_to_enum (System::String^ *name*, System::Int32 *ordinal*)

Add a new enumerated constant to this enum **DDS::TypeCode** (p. 1301).

This method is applicable to **DDS::TypeCode** (p. 1301) objects representing enumerations (DDS::TCKind::TK_ENUM). To add a field to a structured type, see **DDS::TypeCode::add_member_to_enum** (p. 1322).

Modifying a **DDS::TypeCode** (p. 1301) – such as by adding a member – is important if you are using the **Dynamic Data** (p. 73) APIs.

MT Safety:

UNSAFE.

Parameters:

name *<<in>> (p. 175)* The name of the new member. This string must be unique within this type and must not be null.

ordinal *<<in>> (p. 175)* The relative order of the new member in this enum or a custom integer value. The value must be unique within the type.

Returns:

The zero-based index of the new member relative to any other members that previously existed.

See also:

DDS::TypeCode::add_member (p. 1323)
DDS::TypeCode::add_member_ex (p. 1324)
DDS::TypeCodeFactory (p. 1327)

6.221.2.30 System::UInt32 DDS::TypeCode::add_member (System::String^ *name*, System::Int32 *id*, DDS::TypeCode^ *tc*, System::Byte *member_flags*)

Add a new member to this **DDS::TypeCode** (p. 1301).

This method is applicable to **DDS::TypeCode** (p. 1301) objects representing structures (DDS::TCKind::TK_STRUCT), value types (DDS::TCKind::TK_VALUE), sparse value types (DDS::TCKind::TK_SPARSE), and unions (DDS::TCKind::TK_UNION). To add a constant to an enumeration, see **DDS::TypeCode::add_member_to_enum** (p. 1322).

Modifying a **DDS::TypeCode** (p. 1301) – such as by adding a member – is important if you are using the **Dynamic Data** (p. 73) APIs.

Here's a simple code example that adds two fields to a data type, one an integer and another a sequence of integers.

In C#:

```
// Integer:
 myTypeCode.add_member(
     "myFieldName",
     // If the type is sparse, specify an ID. Otherwise, use this sentinel:
     TypeCode.MEMBER_ID_INVALID,
     TypeCodeFactory.get_instance().get_primitive_tc(TCKind.TK_LONG),
     // New field is not a key:
     TypeCode.NONKEY_REQUIRED_MEMBER);

 // Sequence of 10 or fewer integers:
 myTypeCode.add_member(
     "myFieldName",
     // If the type is sparse, specify an ID. Otherwise, use this sentinel:
     TypeCode.MEMBER_ID_INVALID,
     TypeCodeFactory.get_instance().create_sequence_tc(
         10,
         TypeCodeFactory.get_instance().get_primitive_tc(TCKind.TK_LONG)),
     // New field is not a key:
     TypeCode.NONKEY_REQUIRED_MEMBER);
```

In C++/CLI:

```
// Integer:
 myTypeCode->add_member(
     "myFieldName",
     // If the type is sparse, specify an ID. Otherwise, use this sentinel:
     TypeCode::MEMBER_ID_INVALID,
     TypeCodeFactory::get_instance()->get_primitive_tc(TCKind::TK_LONG),
     // New field is not a key:
     TypeCode::NONKEY_REQUIRED_MEMBER);

 // Sequence of 10 or fewer integers:
 myTypeCode->add_member(
     "myFieldName",
```

```
// If the type is sparse, specify an ID. Otherwise, use this sentinel:
TypeCode::MEMBER_ID_INVALID,
TypeCodeFactory::get_instance()->create_sequence_tc(
    10,
    TypeCodeFactory::get_instance()->get_primitive_tc(TCKind::TK_LONG)),
// New field is not a key:
TypeCode::NONKEY_REQUIRED_MEMBER);
```

MT Safety:

UNSAFE.

Parameters:

name *<<in>>* *(p. 175)* The name of the new member.

id *<<in>>* *(p. 175)* The ID of the new member. This should only be specified for members of kind DDS::TCKind::TK_SPARSE and DDS::TCKind::TK_UNION; otherwise, it should be **DDS::TypeCode::MEMBER_ID_INVALID** (p. 65).

tc *<<in>>* *(p. 175)* The type of the new member. You can get or create this **DDS::TypeCode** (p. 1301) with the **DDS::TypeCodeFactory** (p. 1327).

member_flags *<<in>>* *(p. 175)* Indicates whether the member is part of the key and whether it is required.

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. 1324)
DDS::TypeCode::add_member_to_enum (p. 1322)
DDS::TypeCodeFactory (p. 1327)
DDS::TypeCode::NONKEY_MEMBER (p. 65)
DDS::TypeCode::KEY_MEMBER (p. 66)
DDS::TypeCode::NONKEY_REQUIRED_MEMBER (p. 66)

6.221.2.31 System::UInt32 DDS::TypeCode::add_member_ex (System::String^ *name*, System::Int32 *id*, DDS::TypeCode^ *tc*, System::Byte *member_flags*, Visibility *visibility*, System::Boolean *is_pointer*, System::Int16 *bits*)

Add a new member to this **DDS::TypeCode** (p. 1301).

Modifying a **DDS::TypeCode** (p. 1301) – such as by adding a member – is important if you are using the **Dynamic Data** (p. 73) APIs.

MT Safety:

UNSAFE.

Parameters:

name *<<in>>* *(p. 175)* The name of the new member.

id *<<in>>* *(p. 175)* The ID of the new member. This should only be specified for members of kind DDS::TCKind::TK_SPARSE and DDS::TCKind::TK_UNION; otherwise, it should be **DDS::TypeCode::MEMBER_ID_INVALID** (p. 65).

tc *<<in>>* *(p. 175)* The type of the new member. You can get or create this **DDS::TypeCode** (p. 1301) with the **DDS::TypeCodeFactory** (p. 1327).

member_flags *<<in>>* *(p. 175)* Indicates whether the member is part of the key and whether it is required.

visibility *<<in>>* *(p. 175)* Whether the new member is public or private. Non-public members are only relevant for types of kind DDS::TCKind::TK_VALUE and DDS::TCKind::TK_SPARSE.

is_pointer *<<in>>* *(p. 175)* Whether the data member, in its deserialized form, should be stored by pointer as opposed to by value.

bits *<<in>>* *(p. 175)* The number of bits, if this new member is a bit field, or **DDS::TypeCode::NOT_BITFIELD** (p. 65).

Returns:

The zero-based index of the new member relative to any other members that previously existed.

See also:

DDS::TypeCode::add_member (p. 1323)
DDS::TypeCodeFactory (p. 1327)
DDS::TypeCode::NONKEY_MEMBER (p. 65)
DDS::TypeCode::KEY_MEMBER (p. 66)
DDS::TypeCode::NONKEY_REQUIRED_MEMBER (p. 66)

6.221.2.32 void DDS::TypeCode::print_IDL (System::UInt32 *indent*)

Prints a **DDS::TypeCode** (p. 1301) in a pseudo-IDL notation.

MT Safety:

SAFE.

Parameters:

indent *<<**in**>>* *(p. 175)* Indent.

6.221.3 Member Data Documentation

6.221.3.1 TypeCode ^ DDS::TypeCode::TC_NULL

Basic 16-bit signed integer type.

See also:

DDS::TypeCodeFactory::get_primitive_tc (p. 1331)

6.221.3.2 TypeCode ^ DDS::TypeCode::TC_CHAR

Basic single-byte character type.

See also:

DDS::TypeCodeFactory::get_primitive_tc (p. 1331)

6.222 DDS::TypeCodeFactory Class Reference

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

`#include <managed_typecode.h>`

Public Member Functions

- ^ **TypeCode**^ **clone_tc** (**TypeCode**^ tc)

  Creates and returns a copy of the input ***DDS::TypeCode*** *(*p. 1301*).*

- ^ void **delete_tc** (**TypeCode**^ tc)

  Deletes the input ***DDS::TypeCode*** *(*p. 1301*).*

- ^ **TypeCode**^ **get_primitive_tc** (**TCKind** tc_kind)

  Get the ***DDS::TypeCode*** *(*p. 1301*) for a primitive type (integers, floating point values, etc.) identified by the given DDS::TCKind.*

- ^ **TypeCode**^ **create_struct_tc** (System::String^ name, **StructMemberSeq**^ members)

  Constructs a DDS::TCKind::TK_STRUCT ***DDS::TypeCode*** *(*p. 1301*).*

- ^ **TypeCode**^ **create_value_tc** (System::String^ name, **ValueModifier** type_modifier, **TypeCode**^ concrete_base, **ValueMemberSeq**^ members)

  Constructs a DDS::TCKind::TK_VALUE ***DDS::TypeCode*** *(*p. 1301*).*

- ^ **TypeCode**^ **create_union_tc** (System::String^ name, **TypeCode**^ discriminator_type, System::Int32 default_index, **UnionMemberSeq**^ members)

  Constructs a DDS::TCKind::TK_UNION ***DDS::TypeCode*** *(*p. 1301*).*

- ^ **TypeCode**^ **create_enum_tc** (System::String^ name, **EnumMemberSeq**^ members)

  Constructs a DDS::TCKind::TK_ENUM ***DDS::TypeCode*** *(*p. 1301*).*

- ^ **TypeCode**^ **create_alias_tc** (System::String^ name, **TypeCode**^ original_type, System::Boolean is_pointer)

  Constructs a DDS::TCKind::TK_ALIAS (typedef) ***DDS::TypeCode*** *(*p. 1301*).*

- ^ **TypeCode**^ **create_string_tc** (System::UInt32 bound)

Constructs a DDS::TCKind::TK_STRING ***DDS::TypeCode*** *(p. 1301).*

^ **TypeCode**^ **create_wstring_tc** (System::UInt32 bound)

Constructs a DDS::TCKind::TK_WSTRING ***DDS::TypeCode*** *(p. 1301).*

^ **TypeCode**^ **create_sequence_tc** (System::UInt32 bound, **TypeCode**^ element_type)

Constructs a DDS::TCKind::TK_SEQUENCE ***DDS::TypeCode*** *(p. 1301).*

^ **TypeCode**^ **create_array_tc** (**UnsignedIntSeq**^ dimensions, **TypeCode**^ element_type)

Constructs a DDS::TCKind::TK_ARRAY ***DDS::TypeCode*** *(p. 1301).*

^ **TypeCode**^ **create_array_tc** (UInt32 length, **TypeCode**^ element_type)

Constructs a DDS::TCKind::TK_ARRAY ***DDS::TypeCode*** *(p. 1301) for a single-dimensional array.*

^ **TypeCode**^ **create_sparse_tc** (System::String^ name, **ValueModifier** type_modifier, **TypeCode**^ concrete_base)

Constructs a DDS::TCKind::TK_SPARSE ***DDS::TypeCode*** *(p. 1301).*

Static Public Member Functions

^ static **TypeCodeFactory**^ **get_instance** ()

Gets the singleton instance of this class.

6.222.1 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. 1330) method.

If you want to publish and subscribe to data of types that are not known to you at system design time, this class will be your starting point. After creating a data type definition with this class, you will modify that definition using the **DDS::TypeCode** (p. 1301) class and then register it with the **Dynamic Data** (p. 73) API.

The methods of this class fall into several categories:

Getting definitions for primitive types:

Type definitions for primitive types (e.g. integers, floating point values, etc.) are pre-defined; your application only needs to *get* them, not *create* them.

- **DDS::TypeCodeFactory::get_primitive_tc** (p. 1331)

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. 1334)
- **DDS::TypeCodeFactory::create_wstring_tc** (p. 1334)
- **DDS::TypeCodeFactory::create_array_tc** (p. 1335)
- **DDS::TypeCodeFactory::create_array_tc** (p. 1335)
- **DDS::TypeCodeFactory::create_sequence_tc** (p. 1335)

Creating definitions for structured types:

Structured types include structures, value types, sparse value types, and unions.

- **DDS::TypeCodeFactory::create_struct_tc** (p. 1331)
- **DDS::TypeCodeFactory::create_value_tc** (p. 1332)
- **DDS::TypeCodeFactory::create_sparse_tc** (p. 1336)
- **DDS::TypeCodeFactory::create_union_tc** (p. 1332)

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. 1333)
- **DDS::TypeCodeFactory::create_alias_tc** (p. 1333)

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. 1301) that you *created*; if you got the object from **DDS::TypeCodeFactory::get_primitive_tc** (p. 1331), you must *not* delete it.)

- **DDS::TypeCodeFactory::delete_tc** (p. 1330)

Copying type definitions:

You can also create deep copies of type definitions:

- **DDS::TypeCodeFactory::clone_tc** (p. 1330)

6.222.2 Member Function Documentation

6.222.2.1 static TypeCodeFactory ^ DDS::TypeCodeFactory::get_instance () [static]

Gets the singleton instance of this class.

Returns:

The **DDS::TypeCodeFactory** (p. 1327) instance if no errors. Otherwise, null.

6.222.2.2 TypeCode ^ DDS::TypeCodeFactory::clone_tc (TypeCode^ *tc*)

Creates and returns a copy of the input **DDS::TypeCode** (p. 1301).

Parameters:

tc *<<**in**>>* *(p. 175)* Type code that will be copied. Cannot be null.

Returns:

A clone of tc.

6.222.2.3 void DDS::TypeCodeFactory::delete_tc (TypeCode^ *tc*)

Deletes the input **DDS::TypeCode** (p. 1301).

All the type codes created through the **DDS::TypeCodeFactory** (p. 1327) must be deleted using this method.

Parameters:

tc *<<**inout**>>* *(p. 176)* Type code that will be deleted. Cannot be null.

6.222.2.4 TypeCode ^ DDS::TypeCodeFactory::get_primitive_tc (TCKind *tc_kind*)

Get the **DDS::TypeCode** (p. 1301) for a primitive type (integers, floating point values, etc.) identified by the given DDS::TCKind.

See also:

DDS::TypeCode::TC_LONG (p. 63)
DDS::TypeCode::TC_ULONG (p. 63)
DDS::TypeCode::TC_SHORT
DDS::TypeCode::TC_USHORT (p. 63)
DDS::TypeCode::TC_FLOAT (p. 63)
DDS::TypeCode::TC_DOUBLE (p. 64)
DDS::TypeCode::TC_LONGDOUBLE (p. 65)
DDS::TypeCode::TC_OCTET (p. 64)
DDS::TypeCode::TC_BOOLEAN (p. 64)
DDS::TypeCode::TC_CHAR (p. 1326)
DDS::TypeCode::TC_WCHAR (p. 65)

6.222.2.5 TypeCode ^ DDS::TypeCodeFactory::create_struct_tc (System::String^ *name*, StructMemberSeq^ *members*)

Constructs a DDS::TCKind::TK_STRUCT **DDS::TypeCode** (p. 1301).

Parameters:

name *<<in>>* *(p. 175)* Name of the struct type. Cannot be null.

members *<<in>>* *(p. 175)* Initial members of the structure. This list may be empty (that is, **DDS::Sequence::length** (p. 1171) 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.) This argument may be null.

Exceptions:

DDS::ExceptionCode_t::BAD_PARAM_SYSTEM_EXCEPTION_CODE. Illegal parameter value.

Returns:

A newly-created **DDS::TypeCode** (p. 1301) object describing a struct.

6.222.2.6 TypeCode ^ DDS::TypeCodeFactory::create_value_tc (System::String^ *name*, ValueModifier *type_modifier*, TypeCode^ *concrete_base*, ValueMemberSeq^ *members*)

Constructs a DDS::TCKind::TK_VALUE **DDS::TypeCode** (p. 1301).

Parameters:

name *<<in>>* *(p. 175)* Name of the value type. Cannot be null.

type_modifier *<<in>>* *(p. 175)* One of the value type modifier constants: DDS::ValueModifier::VM_NONE, DDS::ValueModifier::VM_CUSTOM, DDS::ValueModifier::VM_ABSTRACT or DDS::ValueModifier::VM_TRUNCATABLE.

concrete_base *<<in>>* *(p. 175)* **DDS::TypeCode** (p. 1301) object describing the concrete valuetype base. It may be null if the valuetype does not have a concrete base.

members *<<in>>* *(p. 175)* 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.) This argument may be null.

Exceptions:

DDS::ExceptionCode_t::BAD_PARAM_SYSTEM_EXCEPTION_CODE. Illegal parameter value.

Returns:

A newly-created **DDS::TypeCode** (p. 1301) object describing a value.

6.222.2.7 TypeCode ^ DDS::TypeCodeFactory::create_union_tc (System::String^ *name*, TypeCode^ *discriminator_type*, System::Int32 *default_index*, UnionMemberSeq^ *members*)

Constructs a DDS::TCKind::TK_UNION **DDS::TypeCode** (p. 1301).

Parameters:

name *<<in>>* *(p. 175)* Name of the union type. Cannot be null.

discriminator_type *<<in>>* *(p. 175)* Discriminator Type Code. Cannot be null.

default_index *<<in>>* *(p. 175)* Index of the default member, or -1 if there is no default member.

members *<<in>>* *(p. 175)* 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.) This argument may be null.

Exceptions:

DDS::ExceptionCode_t::BAD_PARAM_SYSTEM_EXCEPTION_CODE. Illegal parameter value.

Returns:

A newly-created **DDS::TypeCode** (p. 1301) object describing a union.

6.222.2.8 TypeCode ^ DDS::TypeCodeFactory::create_enum_tc (System::String^ *name*, EnumMemberSeq^ *members*)

Constructs a DDS::TCKind::TK_ENUM **DDS::TypeCode** (p. 1301).

Parameters:

name *<<in>>* *(p. 175)* Name of the enum type. Cannot be null.

members *<<in>>* *(p. 175)* 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. 1322). This argument may be null.

Exceptions:

DDS::ExceptionCode_t::BAD_PARAM_SYSTEM_EXCEPTION_CODE. Illegal parameter value.

Returns:

A newly-created **DDS::TypeCode** (p. 1301) object describing an enumeration.

6.222.2.9 TypeCode ^ DDS::TypeCodeFactory::create_alias_tc (System::String^ *name*, TypeCode^ *original_type*, System::Boolean *is_pointer*)

Constructs a DDS::TCKind::TK_ALIAS (typedef) **DDS::TypeCode** (p. 1301).

Parameters:

name *<<in>> (p. 175)* Name of the alias. Cannot be null.

original_type *<<in>> (p. 175)* Aliased type code. Cannot be null.

is_pointer *<<in>> (p. 175)* Indicates if the alias is a pointer to the aliased type code.

Exceptions:

DDS::ExceptionCode_t::BAD_PARAM_SYSTEM_EXCEPTION_CODE. Illegal parameter value.

Returns:

A newly-created **DDS::TypeCode** (p. 1301) object describing an alias.

6.222.2.10 TypeCode ^ DDS::TypeCodeFactory::create_string_tc (System::UInt32 *bound*)

Constructs a DDS::TCKind::TK_STRING **DDS::TypeCode** (p. 1301).

Parameters:

bound *<<in>> (p. 175)* Maximum length of the string.

Exceptions:

DDS::ExceptionCode_t::BAD_PARAM_SYSTEM_EXCEPTION_CODE. Illegal parameter value.

Returns:

A newly-created **DDS::TypeCode** (p. 1301) object describing a string.

6.222.2.11 TypeCode ^ DDS::TypeCodeFactory::create_wstring_tc (System::UInt32 *bound*)

Constructs a DDS::TCKind::TK_WSTRING **DDS::TypeCode** (p. 1301).

Parameters:

bound *<<in>> (p. 175)* Maximum length of the wide string.

Exceptions:

DDS::ExceptionCode_t::BAD_PARAM_SYSTEM_EXCEPTION_CODE. Illegal parameter value.

Returns:

A newly-created **DDS::TypeCode** (p. 1301) object describing a wide string.

6.222.2.12 TypeCode ^ DDS::TypeCodeFactory::create_sequence_tc (System::UInt32 *bound*, TypeCode^ *element_type*)

Constructs a DDS::TCKind::TK_SEQUENCE **DDS::TypeCode** (p. 1301).

Parameters:

bound *<<in>> (p. 175)* The bound for the sequence (> 0).

element_type *<<in>> (p. 175)* **DDS::TypeCode** (p. 1301) object describing the sequence elements.

Exceptions:

DDS::ExceptionCode_t::BAD_PARAM_SYSTEM_EXCEPTION_CODE. Illegal parameter value.

Returns:

A newly-created **DDS::TypeCode** (p. 1301) object describing a sequence.

6.222.2.13 TypeCode ^ DDS::TypeCodeFactory::create_array_tc (UnsignedIntSeq^ *dimensions*, TypeCode^ *element_type*)

Constructs a DDS::TCKind::TK_ARRAY **DDS::TypeCode** (p. 1301).

Parameters:

dimensions *<<in>> (p. 175)* Dimensions of the array. Each dimension has to be greater than 0.

element_type *<<in>> (p. 175)* **DDS::TypeCode** (p. 1301) describing the array elements. Cannot be null.

Exceptions:

DDS::ExceptionCode_t::BAD_PARAM_SYSTEM_EXCEPTION_CODE. Illegal parameter value.

Returns:

A newly-created **DDS::TypeCode** (p. 1301) object describing a sequence.

6.222.2.14 TypeCode ^ DDS::TypeCodeFactory::create_array_tc (UInt32 *length*, TypeCode^ *element_type*)

Constructs a DDS::TCKind::TK_ARRAY **DDS::TypeCode** (p. 1301) for a single-dimensional array.

Parameters:

length *<<in>>* *(p. 175)* Length of the single-dimensional array.

element_type *<<in>>* *(p. 175)* **DDS::TypeCode** (p. 1301) describing the array elements. Cannot be null.

Exceptions:

DDS::ExceptionCode_t::BAD_PARAM_SYSTEM_EXCEPTION_CODE. Illegal parameter value.

Returns:

A newly-created **DDS::TypeCode** (p. 1301) object describing a sequence.

6.222.2.15 TypeCode ^ DDS::TypeCodeFactory::create_sparse_tc (System::String^ *name*, ValueModifier *type_modifier*, TypeCode^ *concrete_base*)

Constructs a DDS::TCKind::TK_SPARSE **DDS::TypeCode** (p. 1301).

A sparse value type is similar to other value types but with one major difference: not all members need to be present in every sample.

It is not possible to generate code for sparse value types; they must be created at runtime using these APIs. You will interact with samples of sparse types using the **Dynamic Data** (p. 73) APIs.

Parameters:

name *<<in>>* *(p. 175)* Name of the value type. Cannot be null.

type_modifier *<<in>>* *(p. 175)* One of the value type modifier constants: DDS::ValueModifier::VM_NONE, DDS::ValueModifier::VM_CUSTOM, DDS::ValueModifier::VM_ABSTRACT or DDS::ValueModifier::VM_TRUNCATABLE.

concrete_base *<<in>>* *(p. 175)* **DDS::TypeCode** (p. 1301) object describing the concrete valuetype base. It may be null if the valuetype does not have a concrete base.

Exceptions:

DDS::ExceptionCode_t::BAD_PARAM_SYSTEM_EXCEPTION_CODE.
Illegal parameter value.

Returns:

A newly-created **DDS::TypeCode** (p. 1301) object describing a value.

6.223 DDS::TypedDataReader< T > Class Template Reference

<<***interface***>> *(*p. *175)* <<***generic***>> *(*p. *175)* User data type-specific data reader.

`#include <managed_subscription.h>`

Inheritance diagram for DDS::TypedDataReader< T >::

Public Member Functions

- ^ void **read** (**DDS::LoanableSequence**< T >^received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, System::UInt32 sample_states, System::UInt32 view_states, System::UInt32 instance_states)

  Access a collection of data samples from the ***DDS::DataReader*** *(*p. *433).*

- ^ void **take** (**DDS::LoanableSequence**< T >^received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, System::UInt32 sample_states, System::UInt32 view_states, System::UInt32 instance_states)

  Access a collection of data-samples from the ***DDS::DataReader*** *(*p. *433).*

- ^ void **read_w_condition** (**DDS::LoanableSequence**< T >^received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, **DDS::ReadCondition**^ condition)

  Accesses via ***DDS::TypedDataReader::read*** *(*p. *1341) the samples that match the criteria specified in the* ***DDS::ReadCondition*** *(*p. *1084).*

- ^ void **take_w_condition** (**DDS::LoanableSequence**< T >^received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, **DDS::ReadCondition**^ condition)

  Analogous to ***DDS::TypedDataReader::read_w_condition*** *(*p. *1349) except it accesses samples via the* ***DDS::TypedDataReader::take*** *(*p. *1342) operation.*

- ^ void **read_next_sample** (T received_data, **DDS::SampleInfo**^ sample_info)

  Copies the next not-previously-accessed data value from the ***DDS::DataReader*** *(*p. *433).*

^ void **take_next_sample** (T received_data, **DDS::SampleInfo**^ sample_info)

*Copies the next not-previously-accessed data value from the **DDS::DataReader*** (p. 433).

^ void **read_instance** (**DDS::LoanableSequence**< T >^received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, **DDS::InstanceHandle_t**% a_handle, System::UInt32 sample_states, System::UInt32 view_states, System::UInt32 instance_states)

*Access a collection of data samples from the **DDS::DataReader*** (p. 433).

^ void **take_instance** (**DDS::LoanableSequence**< T >^received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, **DDS::InstanceHandle_t**% a_handle, System::UInt32 sample_states, System::UInt32 view_states, System::UInt32 instance_states)

*Access a collection of data samples from the **DDS::DataReader*** (p. 433).

^ void **read_next_instance** (**DDS::LoanableSequence**< T >^received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, **DDS::InstanceHandle_t**% previous_handle, System::UInt32 sample_states, System::UInt32 view_states, System::UInt32 instance_states)

*Access a collection of data samples from the **DDS::DataReader*** (p. 433).

^ void **take_next_instance** (**DDS::LoanableSequence**< T >^received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, **DDS::InstanceHandle_t**% previous_handle, System::UInt32 sample_states, System::UInt32 view_states, System::UInt32 instance_states)

*Access a collection of data samples from the **DDS::DataReader*** (p. 433).

^ void **read_next_instance_w_condition** (**DDS::LoanableSequence**< T >^received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, **DDS::InstanceHandle_t**% previous_handle, **DDS::ReadCondition**^ condition)

*Accesses via **DDS::TypedDataReader::read_next_instance*** (p. 1357) *the samples that match the criteria specified in the **DDS::ReadCondition*** (p. 1084).

^ void **take_next_instance_w_condition** (**DDS::LoanableSequence**< T >^received_data, **DDS::SampleInfoSeq**^ info_seq, System::Int32 max_samples, **DDS::InstanceHandle_t**% previous_handle, **DDS::ReadCondition**^ condition)

*Accesses via **DDS::TypedDataReader::take_next_instance*** (p. 1359) *the samples that match the criteria specified in the **DDS::ReadCondition*** (p. 1084).

^ void **return_loan** (**DDS::LoanableSequence**< T >^received_data, **DDS::SampleInfoSeq**^ info_seq)

*Indicates to the **DDS::DataReader** (p. 433) 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. 433).*

^ void **get_key_value** (T key_holder, **DDS::InstanceHandle_t**% handle)

Retrieve the instance `key` *that corresponds to an instance* `handle`*.*

^ **DDS::InstanceHandle_t lookup_instance** (T key_holder)

Retrieve the instance `handle` *that corresponds to an instance* `key_holder`*.*

6.223.1 Detailed Description

template<typename T> class DDS::TypedDataReader< T >

*<<**interface**>> (p. 175) <<**generic**>> (p. 175)* 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. 433)
Foo (p. 877)
DDS::TypedDataWriter (p. 1368)
rtiddsgen (p. 196)

Examples:

HelloWorldSupport.cpp.

6.223.2 Member Function Documentation

6.223.2.1 template<typename T> void DDS::TypedDataReader< T >::read (DDS::LoanableSequence< T >^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, System::UInt32 *sample_states*, System::UInt32 *view_states*, System::UInt32 *instance_states*) [inline]

Access a collection of data samples from the **DDS::DataReader** (p. 433).

This operation offers the same functionality and API as **DDS::TypedDataReader::take** (p. 1342) except that the samples returned remain in the **DDS::DataReader** (p. 433) such that they can be retrieved again by means of a read or take operation.

Please refer to the documentation of **DDS::TypedDataReader::take()** (p. 1342) 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::SampleStateKind::READ_SAMPLE_STATE** (p. 1161). 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::ViewStateKind::NOT_NEW_VIEW_STATE** (p. 1410). It will not affect the `instance_state` of the instance.

Important: If the samples "returned" by this method are loaned from RTI Data Distribution Service (see **DDS::TypedDataReader::take** (p. 1342) for more information on memory loaning), it is important that their contents not be changed. Because the memory in which the data is stored belongs to the middleware, any modifications made to the data will be seen the next time the same samples are read or taken; the samples will no longer reflect the state that was received from the network.

Parameters:

received_data *<<inout>>* *(p. 176)* User data type-specific **DDS::Sequence** (p. 1163) object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 880). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

info_seq *<<inout>>* *(p. 176)* A **DDS::SampleInfoSeq** (p. 1157) object where the received sample info will be returned. Must be a valid non-NULL **DDS::SampleInfoSeq** (p. 1157). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

max_samples *<<in>>* *(p. 175)* The maximum number of samples to

be returned. If the special value DDS::LENGTH_UNLIMITED is provided, as many samples will be returned as are available, up to the limits described in the documentation for **DDS::TypedDataReader::take()** (p. 1342).

sample_states *<<in>>* *(p. 175)* Data samples matching one of these `sample_states` are returned.

view_states *<<in>>* *(p. 175)* Data samples matching one of these `view_state` are returned.

instance_states *<<in>>* *(p. 175)* Data samples matching ones of these `instance_state` are returned.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_PreconditionNotMet** (p. 1123), **DDS::Retcode_NoData** (p. 1120) or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::TypedDataReader::read_w_condition (p. 1349), **DDS::TypedDataReader::take** (p. 1342), **DDS::TypedDataReader::take_w_condition** (p. 1350)
DDS::LENGTH_UNLIMITED

6.223.2.2 template<typename T> void DDS::TypedDataReader< T >::take (DDS::LoanableSequence< T >^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, System::UInt32 *sample_states*, System::UInt32 *view_states*, System::UInt32 *instance_states*) [inline]

Access a collection of data-samples from the **DDS::DataReader** (p. 433).

The operation will return the list of samples received by the **DDS::DataReader** (p. 433) since the last **DDS::TypedDataReader::take** (p. 1342) operation that match the specified DDS::SampleStateMask, DDS::ViewStateMask and DDS::InstanceStateMask.

This operation may fail with **DDS::Retcode_Error** (p. 1116) if **DDS::DataReaderResourceLimitsQosPolicy::max_outstanding_reads** (p. 491) 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. 898), **DDS::ResourceLimitsQosPolicy** (p. 1109), **DDS::DataReaderResourceLimitsQosPolicy** (p. 486) and the

characteristics of the data-type that is associated with the **DDS::DataReader** (p. 433):

- In the case where the **DDS::HistoryQosPolicy::kind** (p. 901) is DDS::HistoryQosPolicyKind::KEEP_LAST_HISTORY_QOS, the call will return at most **DDS::HistoryQosPolicy::depth** (p. 901) samples per instance.

- The maximum number of samples returned is limited by **DDS::ResourceLimitsQosPolicy::max_samples** (p. 1112), and by **DDS::DataReaderResourceLimitsQosPolicy::max_samples_per_read** (p. 492).

- For multiple instances, the number of samples returned is additionally limited by the product (**DDS::ResourceLimitsQosPolicy::max_samples_per_instance** (p. 1113) * **DDS::ResourceLimitsQosPolicy::max_instances** (p. 1112))

- If **DDS::DataReaderResourceLimitsQosPolicy::max_infos** (p. 490) is limited, the number of samples returned may also be limited if insufficient **DDS::SampleInfo** (p. 1148) 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. 1148) 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 insuffificient resources, in order to access remaining samples.

Note that in the case where the **DDS::Topic** (p. 1258) associated with the **DDS::DataReader** (p. 433) is bound to a data-type that has no key definition, then there will be at most one instance in the **DDS::DataReader** (p. 433). So the per-sample limits will apply.

The act of *taking* a sample removes it from RTI Data Distribution Service 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::ViewStateKind::NOT_NEW_VIEW_STATE** (p. 1410). It will not affect the `instance_state` of the sample's instance.

After **DDS::TypedDataReader::take** (p. 1342) 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 **DDS::TypedDataReader::take** (p. 1342) 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. 433). 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. 433) by calling the **DDS::TypedDataReader::return_loan** (p. 1364) 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. 1148) objects after the call to **DDS::TypedDataReader::return_loan** (p. 1364). Returning the loan places the objects back into a pool, allowing the middleware to overwrite them with new data.

Note: While you must call **DDS::TypedDataReader::return_loan** (p. 1364) at some point, you do *not* have to do so before the next **DDS::TypedDataReader::take** (p. 1342) call. However, failure to return the loan will eventually deplete the **DDS::DataReader** (p. 433) 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. 433) is specified by the **DDS::ResourceLimitsQosPolicy** (p. 1109) and the **DDS::DataReaderResourceLimitsQosPolicy** (p. 486).

If the sequences are not empty (maximum size not equal to 0 and length not equal to 0) when **DDS::TypedDataReader::take** (p. 1342) is called, samples are copied to received_data and info_seq. The application will not need to call **DDS::TypedDataReader::return_loan** (p. 1364).

The order of the samples returned to the caller depends on the **DDS::PresentationQosPolicy** (p. 1012).

- If **DDS::PresentationQosPolicy::access_scope** (p. 1015) is DDS::PresentationQosPolicyAccessScopeKind::INSTANCE_PRESENTATION_QOS, the returned collection is a list where samples belonging to the same data instance are consecutive.

- If **DDS::PresentationQosPolicy::access_scope** (p. 1015) is DDS::PresentationQosPolicyAccessScopeKind::TOPIC_PRESENTATION_QOS and **DDS::PresentationQosPolicy::ordered_access** (p. 1016) is set to false, then returned collection is a list where samples belonging to the same data instance are consecutive.

- If **DDS::PresentationQosPolicy::access_scope** (p. 1015) is DDS::PresentationQosPolicyAccessScopeKind::TOPIC_PRESENTATION_QOS and **DDS::PresentationQosPolicy::ordered_access** (p. 1016) is set to true, then the returned collection is a list were the relative order of samples is preserved also accross different instances. Note that samples belonging to the same instance may or may not be

consecutive. This is because to preserve order it may be necessary to mix samples from different instances.

- If **DDS::PresentationQosPolicy::access_scope** (p. 1015) is DDS::PresentationQosPolicyAccessScopeKind::GROUP_PRESENTATION_QOS and **DDS::PresentationQosPolicy::ordered_access** (p. 1016) is set to false, then returned collection is a list where samples belonging to the same data instance are consecutive. [**Not supported (optional)**]

- If **DDS::PresentationQosPolicy::access_scope** (p. 1015) is DDS::PresentationQosPolicyAccessScopeKind::GROUP_PRESENTATION_QOS and **DDS::PresentationQosPolicy::ordered_access** (p. 1016) is set to true, then the returned collection contains at most one sample. The difference in this case is due to the fact that is required that the application is able to read samples belonging to different **DDS::DataReader** (p. 433) objects in a specific order. [**Not supported (optional)**]

In any case, the relative order between the samples of one instance is consistent with the **DESTINATION_ORDER** (p. 292) policy:

- If **DDS::DestinationOrderQosPolicy::kind** (p. 562) is DDS::DestinationOrderQosPolicyKind::BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS, 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. 562) is DDS::DestinationOrderQosPolicyKind::BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS, 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. 433) has no samples that meet the constraints, the method will fail with **DDS::Retcode_NoData** (p. 1120).

In addition to the collection of samples, the read and take operations also use a collection of **DDS::SampleInfo** (p. 1148) structures.

6.223.3 SEQUENCES USAGE IN TAKE AND READ

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 **DDS::Sequence::has_ownership** (p. 1173))
- the current-length (`len`, see **DDS::Sequence::length** (p. 1171))
- the maximum length (`max_len`, see **DDS::Sequence::maximum** (p. 1172))

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_PreconditionNotMet** (p. 1123).
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. 433). 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. 499) using **DDS::TypedDataReader::return_loan** (p. 1364).
4. If the initial `max_len>0` and `owns==FALSE`, then the read or take operation will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123). 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. 1148) values into the elements already inside the collections. On output, `owns` will be TRUE, `len` will be set to the number of values copied and `max_len` will remain unchanged. The use of this variant forces a copy but the application can control where the copy is placed and the application will not need to return the loan. The number of samples copied depends on the relative values of `max_len` and `max_samples`:
   - If `max_samples` == LENGTH_UNLIMITED, then at most `max_len` values will be copied. The use of this variant lets the application limit the number of samples returned to what the sequence can accommodate.

- If `max_samples` <= `max_len`, then at most `max_samples` values will be copied. The use of this variant lets the application limit the number of samples returned to fewer that what the sequence can accommodate.
- If `max_samples` > `max_len`, then the read or take operation will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123). 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. 433), 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. 433). If this is the case, the application will need to use **DDS::TypedDataReader::return_loan** (p. 1364) to return the loan once it is no longer using the `received_data` in the collection. When **DDS::TypedDataReader::return_loan** (p. 1364) 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 **DDS::TypedDataReader::return_loan** (p. 1364), 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. 877) and **DDS::SampleInfo** (p. 1148) 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. 877) values and the collection of **DDS::SampleInfo** (p. 1148) structures are of the same length and are in a one-to-one correspondence. Each **DDS::SampleInfo** (p. 1148) 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. 1148) is **DDS::InstanceStateKind::NOT_ALIVE_DISPOSED_INSTANCE_STATE** (p. 909) or **DDS::InstanceStateKind::NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 909), then the last sample for that instance in the collection (that is, the one whose **DDS::SampleInfo** (p. 1148) 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. 1109). The act of reading/taking a sample sets its sample_state to **DDS::SampleStateKind::READ_SAMPLE_STATE** (p. 1161).

If the sample belongs to the most recent generation of the instance, it will also set the view_state of the instance to **DDS::ViewStateKind::NOT_NEW_VIEW_STATE** (p. 1410). 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. 877)). If the **DDS::DataReader** (p. 433) has no samples that meet the constraints, the operations fails with **DDS::Retcode_NoData** (p. 1120).

For an example on how `take` can be used, please refer to the **receive example** (p. 156).

Parameters:

received_data *<<inout>>* *(p. 176)* User data type-specific **DDS::Sequence** (p. 1163) object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 880). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

Parameters:

info_seq *<<inout>>* *(p. 176)* A **DDS::SampleInfoSeq** (p. 1157) object where the received sample info will be returned. Must be a valid non-NULL **DDS::SampleInfoSeq** (p. 1157). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

max_samples *<<in>>* *(p. 175)* The maximum number of samples to be returned. If the special value DDS::LENGTH_UNLIMITED is provided, as many samples will be returned as are available, up to the limits described above.

sample_states *<<in>>* *(p. 175)* Data samples matching one of these `sample_states` are returned.

view_states *<<in>>* *(p. 175)* Data samples matching one of these `view_state` are returned.

instance_states *<<in>>* *(p. 175)* Data samples matching one of these `instance_state` are returned.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_PreconditionNotMet** (p. 1123), **DDS::Retcode_NoData** (p. 1120) or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::TypedDataReader::read (p. 1341)
DDS::TypedDataReader::read_w_condition (p. 1349),
DDS::TypedDataReader::take_w_condition (p. 1350)
DDS::LENGTH_UNLIMITED

6.223.3.1 template<typename T> void DDS::TypedDataReader< T >::read_w_condition (DDS::LoanableSequence< T >^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, DDS::ReadCondition^ *condition*) [inline]

Accesses via **DDS::TypedDataReader::read** (p. 1341) the samples that match the criteria specified in the **DDS::ReadCondition** (p. 1084).

This operation is especially useful in combination with **DDS::QueryCondition** (p. 1082) to filter data samples based on the content.

The specified **DDS::ReadCondition** (p. 1084) must be attached to the **DDS::DataReader** (p. 433); otherwise the operation will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123).

In case the **DDS::ReadCondition** (p. 1084) is a plain **DDS::ReadCondition** (p. 1084) and not the specialized **DDS::QueryCondition** (p. 1082), the operation is equivalent to calling **DDS::TypedDataReader::read** (p. 1341) 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. 1084).

The samples are accessed with the same semantics as **DDS::TypedDataReader::read** (p. 1341).

If the **DDS::DataReader** (p. 433) has no samples that meet the constraints, the operation will fail with **DDS::Retcode_NoData** (p. 1120).

Parameters:

received_data *<<inout>>* *(p. 176)* user data type-specific **DDS::Sequence** (p. 1163) object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 880). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

info_seq *<<inout>>* *(p. 176)* a **DDS::SampleInfoSeq** (p. 1157) object where the received sample info will be returned. Must be a valid non-NULL **DDS::SampleInfoSeq** (p. 1157). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

max_samples *<<in>>* *(p. 175)* The maximum number of samples to be returned. If the special value DDS::LENGTH_UNLIMITED is provided, as many samples will be returned as are available, up to the limits described in the documentation for **DDS::TypedDataReader::take()** (p. 1342).

condition *<<in>>* *(p. 175)* the **DDS::ReadCondition** (p. 1084) to se-

lect samples of interest. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_PreconditionNotMet** (p. 1123), **DDS::Retcode_NoData** (p. 1120) or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::TypedDataReader::read (p. 1341)
DDS::TypedDataReader::take (p. 1342),
DDS::TypedDataReader::take_w_condition (p. 1350)
DDS::LENGTH_UNLIMITED

6.223.3.2 template<typename T> void DDS::TypedDataReader< T >::take_w_condition (DDS::LoanableSequence< T >^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, DDS::ReadCondition^ *condition*) [inline]

Analogous to **DDS::TypedDataReader::read_w_condition** (p. 1349) except it accesses samples via the **DDS::TypedDataReader::take** (p. 1342) operation.

This operation is analogous to **DDS::TypedDataReader::read_w_condition** (p. 1349) except that it accesses samples via the **DDS::TypedDataReader::take** (p. 1342) operation.

The specified **DDS::ReadCondition** (p. 1084) must be attached to the **DDS::DataReader** (p. 433); otherwise the operation will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123).

The samples are accessed with the same semantics as **DDS::TypedDataReader::take** (p. 1342).

This operation is especially useful in combination with **DDS::QueryCondition** (p. 1082) to filter data samples based on the content.

If the **DDS::DataReader** (p. 433) has no samples that meet the constraints, the method will fail with **DDS::Retcode_NoData** (p. 1120).

Parameters:

received_data *<<inout>>* *(p. 176)* user data type-specific **DDS::Sequence** (p. 1163) object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 880).

The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

info_seq *<<inout>>* *(p. 176)* a **DDS::SampleInfoSeq** (p. 1157) object where the received sample info will be returned. Must be a valid non-NULL **DDS::SampleInfoSeq** (p. 1157). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

max_samples *<<in>>* *(p. 175)* The maximum number of samples to be returned. If the special value DDS::LENGTH_UNLIMITED is provided, as many samples will be returned as are available, up to the limits described in the documentation for **DDS::TypedDataReader::take()** (p. 1342).

condition *<<in>>* *(p. 175)* the **DDS::ReadCondition** (p. 1084) to select samples of interest. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_PreconditionNotMet** (p. 1123), **DDS::Retcode_NoData** (p. 1120) or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::TypedDataReader::read_w_condition (p. 1349),
DDS::TypedDataReader::read (p. 1341)
DDS::TypedDataReader::take (p. 1342)
DDS::LENGTH_UNLIMITED

6.223.3.3 template<typename T> void DDS::TypedDataReader< T >::read_next_sample (T *received_data*, DDS::SampleInfo^ *sample_info*) [inline]

Copies the next not-previously-accessed data value from the **DDS::DataReader** (p. 433).

This operation copies the next not-previously-accessed data value from the **DDS::DataReader** (p. 433). This operation also copies the corresponding **DDS::SampleInfo** (p. 1148). The implied order among the samples stored in the **DDS::DataReader** (p. 433) is the same as for the **DDS::TypedDataReader::read** (p. 1341) operation.

The **DDS::TypedDataReader::read_next_sample** (p. 1351) operation is semantically equivalent to the **DDS::TypedDataReader::read** (p. 1341) 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 **DDS::TypedDataReader::read_next_sample** (p. 1351) 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. 433), the operation will fail with **DDS::Retcode_NoData** (p. 1120) and nothing is copied.

Parameters:

received_data *<<inout>> (p. 176)* user data type-specific **Foo** (p. 877) 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. 877). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

sample_info *<<inout>> (p. 176)* a **DDS::SampleInfo** (p. 1148) object where the next received sample info will be returned. Must be a valid non-NULL **DDS::SampleInfo** (p. 1148). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_NoData** (p. 1120) or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::TypedDataReader::read (p. 1341)

6.223.3.4 template<typename T> void DDS::TypedDataReader< T >::take_next_sample (T *received_data*, DDS::SampleInfo^ *sample_info*) [inline]

Copies the next not-previously-accessed data value from the **DDS::DataReader** (p. 433).

This operation copies the next not-previously-accessed data value from the **DDS::DataReader** (p. 433) and 'removes' it from the **DDS::DataReader** (p. 433) so that it is no longer accessible. This operation also copies the corresponding **DDS::SampleInfo** (p. 1148). This operation is analogous to the **DDS::TypedDataReader::read_next_sample** (p. 1351) except for the fact that the sample is removed from the **DDS::DataReader** (p. 433).

The **DDS::TypedDataReader::take_next_sample** (p. 1352) operation is semantically equivalent to the **DDS::TypedDataReader::take** (p. 1342) 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 **DDS::TypedDataReader::read_next_sample** (p. 1351) 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. 433), the operation will fail with **DDS::Retcode_NoData** (p. 1120) and nothing is copied.

Parameters:

received_data *<<inout>> (p. 176)* user data type-specific **Foo** (p. 877) 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. 877). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

sample_info *<<inout>> (p. 176)* a **DDS::SampleInfo** (p. 1148) object where the next received sample info will be returned. Must be a valid non-NULL **DDS::SampleInfo** (p. 1148). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_NoData** (p. 1120) or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::TypedDataReader::take (p. 1342)

6.223.3.5 template<typename T> void DDS::TypedDataReader< T >::read_instance (DDS::LoanableSequence< T >^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, DDS::InstanceHandle_t% *a_handle*, System::UInt32 *sample_states*, System::UInt32 *view_states*, System::UInt32 *instance_states*) [inline]

Access a collection of data samples from the **DDS::DataReader** (p. 433).

This operation accesses a collection of data values from the **DDS::DataReader** (p. 433). The behavior is identical to **DDS::TypedDataReader::read** (p. 1341), 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. 1148) verifies **DDS::SampleInfo::instance_handle** (p. 1153) == `a_handle`.

The **DDS::TypedDataReader::read_instance** (p. 1353) operation is semantically equivalent to the **DDS::TypedDataReader::read** (p. 1341) operation,

except in building the collection, the **DDS::DataReader** (p. 433) 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 **DDS::TypedDataReader::read_instance** (p. 1353) operation follows the same rules as the **DDS::TypedDataReader::read** (p. 1341) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **DDS::TypedDataReader::read** (p. 1341), the **DDS::TypedDataReader::read_instance** (p. 1353) operation may 'loan' elements to the output collections, which must then be returned by means of **DDS::TypedDataReader::return_loan** (p. 1364).

Similar to the **DDS::TypedDataReader::read** (p. 1341), 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. 433) has no samples that meet the constraints, the method will fail with **DDS::Retcode_NoData** (p. 1120).

This operation may fail with **DDS::Retcode_BadParameter** (p. 1115) if the **DDS::InstanceHandle_t** (p. 905) `a_handle` does not correspond to an existing data-object known to the **DDS::DataReader** (p. 433).

Parameters:

received_data *<<inout>>* *(p. 176)* user data type-specific **DDS::Sequence** (p. 1163) object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 880). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

info_seq *<<inout>>* *(p. 176)* a **DDS::SampleInfoSeq** (p. 1157) object where the received sample info will be returned. Must be a valid non-NULL **DDS::SampleInfoSeq** (p. 1157). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

max_samples *<<in>>* *(p. 175)* The maximum number of samples to be returned. If the special value DDS::LENGTH_UNLIMITED is provided, as many samples will be returned as are available, up to the limits described in the documentation for **DDS::TypedDataReader::take()** (p. 1342).

a_handle *<<in>>* *(p. 175)* The specified instance to return samples for. Must be a valid non-NULL **DDS::InstanceHandle_t** (p. 905). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if the `handle` does not correspond to an existing data-object known to the **DDS::DataReader** (p. 433).

sample_states *<<in>>* *(p. 175)* data samples matching ones of these `sample_states` are returned

view_states <<*in*>> *(p. 175)* data samples matching ones of these `view_state` are returned

instance_states <<*in*>> *(p. 175)* data samples matching ones of these `instance_state` are returned

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_PreconditionNotMet** (p. 1123), **DDS::Retcode_NoData** (p. 1120) or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::TypedDataReader::read (p. 1341)
DDS::LENGTH_UNLIMITED

6.223.3.6 template<typename T> void DDS::TypedDataReader< T >::take_instance (DDS::LoanableSequence< T >^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, DDS::InstanceHandle_t% *a_handle*, System::UInt32 *sample_states*, System::UInt32 *view_states*, System::UInt32 *instance_states*) [inline]

Access a collection of data samples from the **DDS::DataReader** (p. 433).

This operation accesses a collection of data values from the **DDS::DataReader** (p. 433). The behavior is identical to **DDS::TypedDataReader::take** (p. 1342), except for that all samples returned belong to the single specified instance whose handle is `a_handle`.

The semantics are the same for the **DDS::TypedDataReader::take** (p. 1342) operation, except in building the collection, the **DDS::DataReader** (p. 433) 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 **DDS::TypedDataReader::take_instance** (p. 1355) operation follows the same rules as the **DDS::TypedDataReader::read** (p. 1341) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **DDS::TypedDataReader::read** (p. 1341), the **DDS::TypedDataReader::take_instance** (p. 1355) operation may 'loan' elements to the output collections, which must then be returned by means of **DDS::TypedDataReader::return_loan** (p. 1364).

Similar to the **DDS::TypedDataReader::read** (p. 1341), 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. 433) has no samples that meet the constraints, the method fails with **DDS::Retcode_NoData** (p. 1120).

This operation may fail with **DDS::Retcode_BadParameter** (p. 1115) if the **DDS::InstanceHandle_t** (p. 905) `a_handle` does not correspond to an existing data-object known to the **DDS::DataReader** (p. 433).

Parameters:

received_data *<<inout>>* *(p. 176)* user data type-specific **DDS::Sequence** (p. 1163) object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 880). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

info_seq *<<inout>>* *(p. 176)* a **DDS::SampleInfoSeq** (p. 1157) object where the received sample info will be returned. Must be a valid non-NULL **DDS::SampleInfoSeq** (p. 1157). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

max_samples *<<in>>* *(p. 175)* The maximum number of samples to be returned. If the special value DDS::LENGTH_UNLIMITED is provided, as many samples will be returned as are available, up to the limits described in the documentation for **DDS::TypedDataReader::take()** (p. 1342).

a_handle *<<in>>* *(p. 175)* The specified instance to return samples for. Must be a valid non-NULL **DDS::InstanceHandle_t** (p. 905). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL. The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if the `handle` does not correspond to an existing data-object known to the **DDS::DataReader** (p. 433).

sample_states *<<in>>* *(p. 175)* data samples matching ones of these `sample_states` are returned

view_states *<<in>>* *(p. 175)* data samples matching ones of these `view_state` are returned

instance_states *<<in>>* *(p. 175)* data samples matching ones of these `instance_state` are returned

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_PreconditionNotMet** (p. 1123), **DDS::Retcode_NoData** (p. 1120) or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::TypedDataReader::take (p. 1342)
DDS::LENGTH_UNLIMITED

6.223.3.7 template<typename T> void DDS::TypedDataReader< T >::read_next_instance (DDS::LoanableSequence< T >^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, DDS::InstanceHandle_t% *previous_handle*, System::UInt32 *sample_states*, System::UInt32 *view_states*, System::UInt32 *instance_states*) [inline]

Access a collection of data samples from the **DDS::DataReader** (p. 433).

This operation accesses a collection of data values from the **DDS::DataReader** (p. 433) where all the samples belong to a single instance. The behavior is similar to **DDS::TypedDataReader::read_instance** (p. 1353), 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. 433). For the purposes of the ordering, it should be 'as if' each instance handle was represented as unique integer.

The behavior of **DDS::TypedDataReader::read_next_instance** (p. 1357) is 'as if' the **DDS::DataReader** (p. 433) invoked **DDS::TypedDataReader::read_instance** (p. 1353), 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::InstanceHandle_t::HANDLE_NIL** (p. 53) is guaranteed to be 'less than' any valid `instance_handle`. So the use of the parameter value `previous_handle` == **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) will return the samples for the instance which has the smallest `instance_handle` among all the instances that contain available samples.

The operation **DDS::TypedDataReader::read_next_instance** (p. 1357) is intended to be used in an application-driven iteration, where the application starts by passing `previous_handle` == **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), examines the samples returned, and then uses the `instance_handle` returned in the **DDS::SampleInfo** (p. 1148) as the value of the `previous_handle` argument to the next call to **DDS::TypedDataReader::read_next_instance** (p. 1357). The iteration continues until **DDS::TypedDataReader::read_next_instance** (p. 1357)

fails with the value **DDS::Retcode_NoData** (p. 1120).

Note that it is possible to call the **DDS::TypedDataReader::read_next_instance** (p. 1357) operation with a `previous_handle` that does not correspond to an instance currently managed by the **DDS::DataReader** (p. 433). This is because as stated earlier the 'greater-than' relationship is defined even for handles not managed by the **DDS::DataReader** (p. 433). One practical situation where this may occur is when an application is iterating though all the instances, takes all the samples of a **DDS::InstanceStateKind::NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 909) 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 **DDS::TypedDataReader::read_next_instance** (p. 1357) operation follows the same rules as the **DDS::TypedDataReader::read** (p. 1341) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **DDS::TypedDataReader::read** (p. 1341), the **DDS::TypedDataReader::read_instance** (p. 1353) operation may 'loan' elements to the output collections, which must then be returned by means of **DDS::TypedDataReader::return_loan** (p. 1364).

Similar to the **DDS::TypedDataReader::read** (p. 1341), 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. 433) has no samples that meet the constraints, the method will fail with **DDS::Retcode_NoData** (p. 1120).

Parameters:

received_data *<<inout>>* *(p. 176)* user data type-specific **DDS::Sequence** (p. 1163) object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 880). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

info_seq *<<inout>>* *(p. 176)* a **DDS::SampleInfoSeq** (p. 1157) object where the received sample info will be returned. Must be a valid non-NULL **DDS::SampleInfoSeq** (p. 1157). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

max_samples *<<in>>* *(p. 175)* The maximum number of samples to be returned. If the special value DDS::LENGTH_UNLIMITED is provided, as many samples will be returned as are available, up to the limits described in the documentation for **DDS::TypedDataReader::take()** (p. 1342).

previous_handle *<<in>>* *(p. 175)* 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. 905). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

sample_states <<*in*>> *(p. 175)* data samples matching ones of these `sample_states` are returned

view_states <<*in*>> *(p. 175)* data samples matching ones of these `view_state` are returned

instance_states <<*in*>> *(p. 175)* data samples matching ones of these `instance_state` are returned

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_PreconditionNotMet** (p. 1123) **DDS::Retcode_NoData** (p. 1120) or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::TypedDataReader::read (p. 1341)
DDS::LENGTH_UNLIMITED

6.223.3.8 template<typename T> void DDS::TypedDataReader< T >::take_next_instance (DDS::LoanableSequence< T >^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, DDS::InstanceHandle_t% *previous_handle*, System::UInt32 *sample_states*, System::UInt32 *view_states*, System::UInt32 *instance_states*) [inline]

Access a collection of data samples from the **DDS::DataReader** (p. 433).

This operation accesses a collection of data values from the **DDS::DataReader** (p. 433) and 'removes' them from the **DDS::DataReader** (p. 433).

This operation has the same behavior as **DDS::TypedDataReader::read_next_instance** (p. 1357), except that the samples are 'taken' from the **DDS::DataReader** (p. 433) such that they are no longer accessible via subsequent 'read' or 'take' operations.

Similar to the operation **DDS::TypedDataReader::read_next_instance** (p. 1357), it is possible to call **DDS::TypedDataReader::take_next_instance** (p. 1359) with a `previous_handle` that does not correspond to an instance currently managed by the **DDS::DataReader** (p. 433).

The behavior of the **DDS::TypedDataReader::take_next_instance** (p. 1359) operation follows the same rules as the

DDS::TypedDataReader::read (p. 1341) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **DDS::TypedDataReader::read** (p. 1341), the **DDS::TypedDataReader::take_next_instance** (p. 1359) operation may 'loan' elements to the output collections, which must then be returned by means of **DDS::TypedDataReader::return_loan** (p. 1364).

Similar to the **DDS::TypedDataReader::read** (p. 1341), 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. 433) has no samples that meet the constraints, the method will fail with **DDS::Retcode_NoData** (p. 1120).

Parameters:

received_data *<<inout>>* *(p. 176)* user data type-specific **DDS::Sequence** (p. 1163) object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 880). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

info_seq *<<inout>>* *(p. 176)* a **DDS::SampleInfoSeq** (p. 1157) object where the received sample info will be returned. Must be a valid non-NULL **DDS::SampleInfoSeq** (p. 1157). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

max_samples *<<in>>* *(p. 175)* The maximum number of samples to be returned. If the special value DDS::LENGTH_UNLIMITED is provided, as many samples will be returned as are available, up to the limits described in the documentation for **DDS::TypedDataReader::take()** (p. 1342).

previous_handle *<<in>>* *(p. 175)* 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. 905). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

sample_states *<<in>>* *(p. 175)* data samples matching ones of these `sample_states` are returned

view_states *<<in>>* *(p. 175)* data samples matching ones of these `view_state` are returned

instance_states *<<in>>* *(p. 175)* data samples matching ones of these `instance_state` are returned

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_PreconditionNotMet** (p. 1123), **DDS::Retcode_NoData** (p. 1120) or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::TypedDataReader::take (p. 1342)
DDS::LENGTH_UNLIMITED

6.223.3.9 template<typename T> void DDS::TypedDataReader< T >::read_next_instance_w_condition (DDS::LoanableSequence< T >^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, DDS::InstanceHandle_t% *previous_handle*, DDS::ReadCondition^ *condition*) [inline]

Accesses via **DDS::TypedDataReader::read_next_instance** (p. 1357) the samples that match the criteria specified in the **DDS::ReadCondition** (p. 1084).

This operation access a collection of data values from the **DDS::DataReader** (p. 433). The behavior is identical to **DDS::TypedDataReader::read_next_instance** (p. 1357), except that all samples returned 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. 1084) evaluates to TRUE.

Similar to the operation **DDS::TypedDataReader::read_next_instance** (p. 1357), it is possible to call **DDS::TypedDataReader::read_next_instance_w_condition** (p. 1361) with a `previous_handle` that does not correspond to an instance currently managed by the **DDS::DataReader** (p. 433).

The behavior of the **DDS::TypedDataReader::read_next_instance_w_condition** (p. 1361) operation follows the same rules as the **DDS::TypedDataReader::read** (p. 1341) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **DDS::TypedDataReader::read** (p. 1341), the **DDS::TypedDataReader::read_next_instance_w_condition** (p. 1361) operation may 'loan' elements to the output collections, which must then be returned by means of **DDS::TypedDataReader::return_loan** (p. 1364).

Similar to the **DDS::TypedDataReader::read** (p. 1341), 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. 433) has no samples that meet the constraints, the method will fail with **DDS::Retcode_NoData** (p. 1120).

Parameters:

received_data *<<inout>>* *(p. 176)* user data type-specific

DDS::Sequence (p. 1163) object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 880). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

info_seq *<<inout>> (p. 176)* a **DDS::SampleInfoSeq** (p. 1157) object where the received sample info will be returned. Must be a valid non-NULL **DDS::SampleInfoSeq** (p. 1157). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

max_samples *<<in>> (p. 175)* The maximum number of samples to be returned. If the special value DDS::LENGTH_UNLIMITED is provided, as many samples will be returned as are available, up to the limits described in the documentation for **DDS::TypedDataReader::take()** (p. 1342).

previous_handle *<<in>> (p. 175)* 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. 905). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

condition *<<in>> (p. 175)* the **DDS::ReadCondition** (p. 1084) to select samples of interest. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_PreconditionNotMet** (p. 1123) **DDS::Retcode_NoData** (p. 1120) or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::TypedDataReader::read_next_instance (p. 1357)
DDS::LENGTH_UNLIMITED

6.223.3.10 template<typename T> void DDS::TypedDataReader< T >::take_next_instance_w_condition (DDS::LoanableSequence< T >^ *received_data*, DDS::SampleInfoSeq^ *info_seq*, System::Int32 *max_samples*, DDS::InstanceHandle_t% *previous_handle*, DDS::ReadCondition^ *condition*) [inline]

Accesses via **DDS::TypedDataReader::take_next_instance** (p. 1359) the samples that match the criteria specified in the **DDS::ReadCondition** (p. 1084).

This operation access a collection of data values from the **DDS::DataReader** (p. 433) and 'removes' them from the **DDS::DataReader** (p. 433).

The operation has the same behavior as **DDS::TypedDataReader::read_next_instance_w_condition** (p. 1361), except that the samples are 'taken' from the **DDS::DataReader** (p. 433) such that they are no longer accessible via subsequent 'read' or 'take' operations.

Similar to the operation **DDS::TypedDataReader::read_next_instance** (p. 1357), it is possible to call **DDS::TypedDataReader::take_next_instance_w_condition** (p. 1362) with a `previous_handle` that does not correspond to an instance currently managed by the **DDS::DataReader** (p. 433).

The behavior of the **DDS::TypedDataReader::take_next_instance_w_condition** (p. 1362) operation follows the same rules as the **DDS::TypedDataReader::read** (p. 1341) operation regarding the pre-conditions and post-conditions for the `received_data` and `sample_info`. Similar to the **DDS::TypedDataReader::read** (p. 1341), the **DDS::TypedDataReader::take_next_instance_w_condition** (p. 1362) operation may 'loan' elements to the output collections, which must then be returned by means of **DDS::TypedDataReader::return_loan** (p. 1364).

Similar to the **DDS::TypedDataReader::read** (p. 1341), 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. 433) has no samples that meet the constraints, the method will fail with **DDS::Retcode_NoData** (p. 1120).

Parameters:

received_data *<<inout>>* *(p. 176)* user data type-specific **DDS::Sequence** (p. 1163) object where the received data samples will be returned. Must be a valid non-NULL **FooSeq** (p. 880). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

info_seq *<<inout>>* *(p. 176)* a **DDS::SampleInfoSeq** (p. 1157) object where the received sample info will be returned. Must be a valid non-NULL **DDS::SampleInfoSeq** (p. 1157). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

max_samples *<<in>>* *(p. 175)* The maximum number of samples to be returned. If the special value DDS::LENGTH_UNLIMITED is provided, as many samples will be returned as are available, up to the limits described in the documentation for **DDS::TypedDataReader::take()** (p. 1342).

previous_handle *<<in>>* *(p. 175)* 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. 905). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

condition *<<in>>* *(p. 175)* the **DDS::ReadCondition** (p. 1084) to select samples of interest. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_PreconditionNotMet** (p. 1123), or **DDS::Retcode_NoData** (p. 1120), **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::TypedDataReader::take_next_instance (p. 1359)
DDS::LENGTH_UNLIMITED

6.223.3.11 template<typename T> void DDS::TypedDataReader< T >::return_loan (DDS::LoanableSequence< T >^ *received_data*, DDS::SampleInfoSeq^ *info_seq*) [inline]

Indicates to the **DDS::DataReader** (p. 433) 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. 433).

This operation indicates to the **DDS::DataReader** (p. 433) 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. 433).

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. 433) to which they are returned. If either of these conditions is not met, the operation will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123).

The operation **DDS::TypedDataReader::return_loan** (p. 1364) allows implementations of the read and take operations to "loan" buffers from the **DDS::DataReader** (p. 433) to the application and in this manner provide "zero-copy" access to the data. During the loan, the **DDS::DataReader** (p. 433) 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. 433), the application should not retain them indefinitely.

The use of **DDS::TypedDataReader::return_loan** (p. 1364) 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 **DDS::TypedDataReader::return_loan** (p. 1364) 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 **DDS::TypedDataReader::return_loan** (p. 1364), the collections will have `max_len=0`.

Similar to read, this operation must be provided on the specialized class that is generated for the particular application data-type that is being taken.

Parameters:

received_data *<<in>>* *(p. 175)* user data type-specific **DDS::Sequence** (p. 1163) object where the received data samples was obtained from earlier invocation of read or take on the **DDS::DataReader** (p. 433). Must be a valid non-NULL **FooSeq** (p. 880). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

Parameters:

info_seq *<<in>>* *(p. 175)* a **DDS::SampleInfoSeq** (p. 1157) object where the received sample info was obtained from earlier invocation of read or take on the **DDS::DataReader** (p. 433). Must be a valid non-NULL **DDS::SampleInfoSeq** (p. 1157). The method will fail with **DDS::Retcode_BadParameter** (p. 1115) if it is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_PreconditionNotMet** (p. 1123) or **DDS::Retcode_NotEnabled** (p. 1121).

6.223.3.12 template<typename T> void DDS::TypedDataReader< T >::get_key_value (T *key_holder*, DDS::InstanceHandle_t% *handle*) [inline]

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.

For keyed data types, this operation may fail with **DDS::Retcode_-BadParameter** (p. 1115) if the `handle` does not correspond to an existing data-object known to the **DDS::DataReader** (p. 433).

Parameters:

key_holder *<<inout>>* *(p. 176)* a user data type specific key holder, whose `key` fields are filled by this operation. If **Foo** (p. 877) has no key, this method has no effect. This method will fail with **DDS::Retcode_BadParameter** (p. 1115) if `key_holder` is NULL.

handle *<<in>>* *(p. 175)* the `instance` whose key is to be retrieved. If **Foo** (p. 877) *has* a key, `handle` must represent an existing instance of type **Foo** (p. 877) known to the **DDS::DataReader** (p. 433). Otherwise, this method will fail with **DDS::Retcode_-BadParameter** (p. 1115). If **Foo** (p. 877) *has* a key and `handle` is **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), this method will fail with **DDS::Retcode_BadParameter** (p. 1115). If **Foo** (p. 877) has a key and `handle` represents an instance of another type or an instance of type **Foo** (p. 877) that has been unregistered, this method will fail with **DDS::Retcode_BadParameter** (p. 1115). If **Foo** (p. 877) has no key, this method has no effect. This method will fail with **DDS::Retcode_BadParameter** (p. 1115) if `handle` is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-NotEnabled** (p. 1121).

See also:

DDS::TypedDataWriter::get_key_value (p. 1383)

6.223.3.13 template<typename T> DDS::InstanceHandle_t DDS::TypedDataReader< T >::lookup_instance (T *key_holder*) [inline]

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 the Service is unable to provide an instance handle, the Service will return the special value HANDLE_NIL.

Parameters:

key_holder *<<in>>* *(*p. *175)* a user data type specific key holder.

Returns:

the instance handle associated with this instance. If **Foo** (p. 877) has no key, this method has no effect and returns **DDS::InstanceHandle_-t::HANDLE_NIL** (p. 53)

6.224 DDS::TypedDataWriter< T > Class Template Reference

<<***interface***>> *(*p. *175)* <<***generic***>> *(*p. *175)* User data type specific data writer.

`#include <managed_publication.h>`

Inheritance diagram for DDS::TypedDataWriter< T >::

Public Member Functions

- **InstanceHandle_t register_instance** (T instance_data)

  Informs RTI Data Distribution Service that the application will be modifying a particular instance.

- **InstanceHandle_t register_instance_w_timestamp** (T instance_data, **DDS::Time_t**% source_timestamp)

  Performs the same functions as register_instance except that the application provides the value for the `source_timestamp`*.*

- void **unregister_instance** (T instance_data, **DDS::InstanceHandle_t**% handle)

  Reverses the action of ***DDS::TypedDataWriter::register_instance*** *(*p. *1370).*

- void **unregister_instance_w_timestamp** (T instance_data, **DDS::InstanceHandle_t**% handle, **DDS::Time_t**% source_timestamp)

  Performs the same function as ***DDS::TypedDataWriter::unregister_instance*** *(*p. *1372) except that it also provides the value for the* `source_timestamp`*.*

- void **write** (T instance_data, **DDS::InstanceHandle_t**% handle)

  Modifies the value of a data instance.

- void **write_w_timestamp** (T instance_data, **DDS::InstanceHandle_t**% handle, **DDS::Time_t**% source_timestamp)

  Performs the same function as ***DDS::TypedDataWriter::write*** *(*p. *1376) except that it also provides the value for the* `source_timestamp`*.*

- void **dispose** (T instance_data, **DDS::InstanceHandle_t**% instance_handle)

Requests the middleware to delete the data.

^ void **dispose_w_timestamp** (T instance_data, **DDS::InstanceHandle_t**% instance_handle, **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. 433) objects by means of the* `source_timestamp` *attribute inside the* ***DDS::SampleInfo*** *(p. 1148).*

^ void **get_key_value** (T key_holder, **DDS::InstanceHandle_t**% handle)

Retrieve the instance `key` *that corresponds to an instance* `handle`*.*

^ **InstanceHandle_t lookup_instance** (T key_holder)

Retrieve the instance `handle` *that corresponds to an instance* `key_holder`*.*

6.224.1 Detailed Description

template<typename T> class DDS::TypedDataWriter< T >

<<***interface***>> *(p. 175)* <<***generic***>> *(p. 175)* 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. 499)
Foo (p. 877)
DDS::TypedDataReader (p. 1338)
rtiddsgen (p. 196)

Examples:

HelloWorldSupport.cpp.

6.224.2 Member Function Documentation

6.224.2.1 template<typename T> InstanceHandle_t DDS::TypedDataWriter< T >::register_instance (T *instance_data*) [inline]

Informs RTI Data Distribution Service 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::InstanceHandle_t::HANDLE_NIL** (p. 53). The operation takes as a parameter an instance (of which only the key value is examined) and returns a `handle` that can be used in successive **write()** (p. 1376) or **dispose()** (p. 1379) operations.

The operation gives RTI Data Distribution Service 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::InstanceHandle_t::HANDLE_NIL** (p. 53) as the **DDS::InstanceHandle_t** (p. 905) paramater to the write or dispose operation and RTI Data Distribution Service will auto-register the instance.

For best performance, the operation should be invoked prior to calling any operation that modifies the instance, such as **DDS::TypedDataWriter::write** (p. 1376), **DDS::TypedDataWriter::write_w_timestamp** (p. 1378), **DDS::TypedDataWriter::dispose** (p. 1379) and **DDS::TypedDataWriter::dispose_w_timestamp** (p. 1381) and the handle used in conjunction with the data for those calls.

When this operation is used, RTI Data Distribution Service will automatically supply the value of the `source_timestamp` that is used.

This operation may fail and return **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) if **DDS::ResourceLimitsQosPolicy::max_instances** (p. 1112) 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. 499) has been enabled. Otherwise, **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) will be returned.

Parameters:

instance_data *<<in>> (p. 175)* 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. 905), such as write, dispose, unregister_instance, or return **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) on failure. If the instance_data is of a data type that has no keys, this function always return **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53).

See also:

DDS::TypedDataWriter::unregister_instance (p. 1372), **DDS::TypedDataWriter::get_key_value** (p. 1383), **RELATIONSHIP BETWEEN REGISTRATION, LIVELINESS and OWNERSHIP** (p. 995)

6.224.2.2 template<typename T> InstanceHandle_t DDS::TypedDataWriter< T >::register_instance_w_timestamp (T *instance_data*, DDS::Time_t% *source_timestamp*) [inline]

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. 292) QoS policy for details.

This operation may fail and return **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) if **DDS::ResourceLimitsQosPolicy::max_instances** (p. 1112) limit has been exceeded.

This operation can only be called after **DDS::DataWriter** (p. 499) has been enabled. Otherwise, **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) will be returned.

Parameters:

instance_data *<<in>>* *(p. 175)* The instance that should be registered. Of this instance, only the fields that represent the key are examined by the function. Cannot be NULL.

source_timestamp *<<in>>* *(p. 175)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation (used in a *register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application provided timestamp). This timestamp may potentially affect the order in which readers observe events from multiple writers. 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. 905), such as write, dispose, unregister_instance, or return **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) on failure. If the instance_data is of a data type that has no keys, this function always return **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53).

See also:

DDS::TypedDataWriter::unregister_instance (p. 1372), **DDS::TypedDataWriter::get_key_value** (p. 1383)

6.224.2.3 template<typename T> void DDS::TypedDataWriter< T >::unregister_instance (T *instance_data*, DDS::InstanceHandle_t% *handle*) [inline]

Reverses the action of **DDS::TypedDataWriter::register_instance** (p. 1370).

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 **DDS::TypedDataWriter::register_instance** (p. 1370). Otherwise, this operation may fail with **DDS::Retcode_BadParameter** (p. 1115).

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 Data Distribution Service will automatically supply the value of the `source_timestamp` that is used.

This operation informs RTI Data Distribution Service that the **DDS::DataWriter** (p. 499) is no longer going to provide any information about the instance. This operation also indicates that RTI Data Distribution Service can locally remove all information regarding that instance. The application should not attempt to use the `handle` previously allocated to that instance after calling **DDS::TypedDataWriter::unregister_instance()** (p. 1372).

The special value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) 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::InstanceHandle_t::HANDLE_NIL** (p. 53), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with **DDS::Retcode_BadParameter** (p. 1115).

RTI Data Distribution Service will not detect the error when the `handle` is any value other than **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), 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 Data Distribution Service will treat as if the **unregister_instance()** (p. 1372) operation is for the instance as indicated by the `handle`.

If after a **DDS::TypedDataWriter::unregister_instance** (p. 1372), the application wants to modify (**DDS::TypedDataWriter::write** (p. 1376) or **DDS::TypedDataWriter::dispose** (p. 1379)) an instance, it has to register it again, or else use the special `handle` value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53).

This operation does not indicate that the instance is deleted (that is the purpose of **DDS::TypedDataWriter::dispose** (p. 1379)). The operation **DDS::TypedDataWriter::unregister_instance** (p. 1372) just indicates that the **DDS::DataWriter** (p. 499) no longer has anything to say about the instance. **DDS::DataReader** (p. 433) entities that are reading the instance may receive a sample with **DDS::InstanceStateKind::NOT_ALIVE_NO_WRITERS_INSTANCE_STATE** (p. 909) for the instance, unless there are other **DDS::DataWriter** (p. 499) objects writing that same instance.

This operation can affect the ownership of the data instance (see **OWNERSHIP** (p. 283)). If the **DDS::DataWriter** (p. 499) was the exclusive owner of the instance, then calling **unregister_instance()** (p. 1372) will relinquish that ownership.

If **DDS::ReliabilityQosPolicy::kind** (p. 1097) is set to DDS::ReliabilityQosPolicyKind::RELIABLE_RELIABILITY_QOS 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. 1097); if this writer is still unable to unregister after that period, this method will fail with **DDS::Retcode_Timeout** (p. 1124).

Parameters:

instance_data *<<in>>* *(p. 175)* The instance that should be unregistered. If **Foo** (p. 877) *has* a key and `instance_handle` is **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used. If `instance_data` is used, it must represent an instance that has been registerd. Otherwise, this method may fail with **DDS::Retcode_BadParameter** (p. 1115) . If **Foo** (p. 877) *has* a key, `instance_data` can be NULL only if `handle` is not

DDS::InstanceHandle_t::HANDLE_NIL (p. 53). Otherwise, this method will fail with **DDS::Retcode_BadParameter** (p. 1115).

handle *<<**in**>>* *(p. 175)* represents the `instance` to be unregistered. If **Foo** (p. 877) *has* a key and `handle` is **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), `handle` is not used and `instance` is deduced from `instance_data`. If **Foo** (p. 877) has no key, `handle` is not used. If `handle` is used, it must represent an instance that has been registered. Otherwise, this method may fail with **DDS::Retcode_BadParameter** (p. 1115). This method will fail with **DDS::Retcode_BadParameter** (p. 1115) if `handle` is NULL. If **Foo** (p. 877) *has* a key, `handle` cannot be **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) if `instance_data` is NULL. Otherwise, this method will report the error **DDS::Retcode_BadParameter** (p. 1115).

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_Timeout** (p. 1124) or **DDS::Retcode_NotEnabled** (p. 1121)

See also:

DDS::TypedDataWriter::register_instance (p. 1370)
DDS::TypedDataWriter::unregister_instance_w_timestamp (p. 1374)
DDS::TypedDataWriter::get_key_value (p. 1383)
RELATIONSHIP BETWEEN REGISTRATION, LIVELINESS and OWNERSHIP (p. 995)

6.224.2.4 template<typename T> void DDS::TypedDataWriter< T >::unregister_instance_w_timestamp (T *instance_data*, DDS::InstanceHandle_t% *handle*, DDS::Time_t% *source_timestamp*) [inline]

Performs the same function as **DDS::TypedDataWriter::unregister_instance** (p. 1372) 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. 292) QoS policy for details.

The constraints on the values of the `handle` parameter and the corresponding error behavior are the same specified for the **DDS::TypedDataWriter::unregister_instance** (p. 1372) operation.

This operation may block and may time out (**DDS::Retcode_Timeout** (p. 1124)) under the same circumtances described for the unregister_instance operation.

Parameters:

instance_data *<<in>>* *(p. 175)* The instance that should be unregistered. If **Foo** (p. 877) *has* a key and `instance_handle` is **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used. If `instance_data` is used, it must represent an instance that has been registerd. Otherwise, this method may fail with **DDS::Retcode_BadParameter** (p. 1115). If **Foo** (p. 877) *has* a key, `instance_data` can be NULL only if `handle` is not **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53). Otherwise, this method will fail with **DDS::Retcode_BadParameter** (p. 1115).

handle *<<in>>* *(p. 175)* represents the `instance` to be unregistered. If **Foo** (p. 877) *has* a key and `handle` is **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), `handle` is not used and `instance` is deduced from `instance_data`. If **Foo** (p. 877) has no key, `handle` is not used. If `handle` is used, it must represent an instance that has been registered. Otherwise, this method may fail with **DDS::Retcode_BadParameter** (p. 1115). This method will fail with **DDS::Retcode_BadParameter** (p. 1115) if `handle` is NULL. If **Foo** (p. 877) *has* a key, `handle` cannot be **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) if `instance_data` is NULL. Otherwise, this method will fail with **DDS::Retcode_BadParameter** (p. 1115).

source_timestamp *<<in>>* *(p. 175)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation (used in a *register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application provided timestamp). This timestamp may potentially affect the order in which readers observe events from multiple writers. Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_Timeout** (p. 1124) or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::TypedDataWriter::register_instance (p. 1370)
DDS::TypedDataWriter::unregister_instance (p. 1372)
DDS::TypedDataWriter::get_key_value (p. 1383)

6.224.2.5 template<typename T> void DDS::TypedDataWriter< T >::write (T *instance_data*, DDS::InstanceHandle_t% *handle*) [inline]

Modifies the value of a data instance.

When this operation is used, RTI Data Distribution Service will automatically supply the value of the `source_timestamp` that is made available to **DDS::DataReader** (p. 433) objects by means of the `source_timestamp` attribute inside the **DDS::SampleInfo** (p. 1148). (Refer to **DDS::SampleInfo** (p. 1148) and **DESTINATION_ORDER** (p. 292) QoS policy for details).

As a side effect, this operation asserts liveliness on the **DDS::DataWriter** (p. 499) itself, the **DDS::Publisher** (p. 1044) and the **DDS::DomainParticipant** (p. 577).

Note that the special value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) 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::InstanceHandle_t::HANDLE_NIL** (p. 53), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with **DDS::Retcode_BadParameter** (p. 1115).

RTI Data Distribution Service will not detect the error when the `handle` is any value other than **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), 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 Data Distribution Service will treat as if the **write()** (p. 1376) operation is for the instance as indicated by the `handle`.

This operation may block if the **RELIABILITY** (p. 290) `kind` is set to DDS::ReliabilityQosPolicyKind::RELIABLE_RELIABILITY_QOS and the modification would cause data to be lost or else cause one of the limits specified in the **RESOURCE_LIMITS** (p. 298) to be exceeded.

Specifically, this operation may block in the following situations (note that the list may not be exhaustive), even if its DDS::HistoryQosPolicyKind is DDS::HistoryQosPolicyKind::KEEP_LAST_HISTORY_QOS:

- If (**DDS::ResourceLimitsQosPolicy::max_samples** (p. 1112) < **DDS::ResourceLimitsQosPolicy::max_instances** (p. 1112) * **DDS::HistoryQosPolicy::depth** (p. 901)), then in the situation where the `max_samples` resource limit is exhausted, RTI Data Distribution Service is allowed to discard samples of some other instance, as long as at least one sample remains for such an instance. If it is still not possible to make space available to store the modification, the writer is

allowed to block.

- If (**DDS::ResourceLimitsQosPolicy::max_samples** (p. 1112) < **DDS::ResourceLimitsQosPolicy::max_instances** (p. 1112)), then the **DataWriter** (p. 499) may block regardless of the **DDS::HistoryQosPolicy::depth** (p. 901).

This operation may also block when using DDS::ReliabilityQosPolicyKind::BEST_EFFORT_RELIABILITY_QOS and DDS::PublishModeQosPolicyKind::ASYNCHRONOUS_PUBLISH_MODE_QOS. In this case, the **DDS::DataWriter** (p. 499) will queue samples until they are sent by the asynchronous publishing thread. The number of samples that can be stored is determined by the **DDS::HistoryQosPolicy** (p. 898). If the asynchronous thread does not send samples fast enough (e.g., when using a slow **DDS::FlowController** (p. 867)), the queue may fill up. In that case, subsequent write calls will block.

If this operation *does* block for any of the above reasons, the **RELIABILITY** (p. 290) `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** (p. 499) is able to store the modification without exceeding the limits, the operation will time out (**DDS::Retcode_Timeout** (p. 1124)).

If there are no instance resources left, this operation may fail with **DDS::Retcode_OutOfResources** (p. 1122). Calling **DDS::TypedDataWriter::unregister_instance** (p. 1372) may help freeing up some resources.

This operation will fail with **DDS::Retcode_PreconditionNotMet** (p. 1123) 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).

Parameters:

instance_data *<<in>>* *(p. 175)* The data to write.

This method will fail with **DDS::Retcode_BadParameter** (p. 1115) if `instance_data` is NULL.

Parameters:

handle *<<in>>* *(p. 175)* Either the handle returned by a previous call to **DDS::TypedDataWriter::register_instance** (p. 1370), or else the special value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53). If **Foo** (p. 877) *has* a key and `handle` is not **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), `handle` must represent a registered instance of type **Foo** (p. 877). Otherwise, this method may fail with

DDS::Retcode_BadParameter (p. 1115). This method will fail with **DDS::Retcode_BadParameter** (p. 1115) if `handle` is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_Timeout** (p. 1124), **DDS::Retcode_PreconditionNotMet** (p. 1123), **DDS::Retcode_OutOfResources** (p. 1122), or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::DataReader (p. 433)
DDS::TypedDataWriter::write_w_timestamp (p. 1378)
DESTINATION_ORDER (p. 292)

6.224.2.6 template<typename T> void DDS::TypedDataWriter< T >::write_w_timestamp (T *instance_data*, DDS::InstanceHandle_t% *handle*, DDS::Time_t% *source_timestamp*) [inline]

Performs the same function as **DDS::TypedDataWriter::write** (p. 1376) except that it also provides the value for the `source_timestamp`.

Explicitly provides the timestamp that will be available to the **DDS::DataReader** (p. 433) objects by means of the `source_timestamp` attribute inside the **DDS::SampleInfo** (p. 1148). (Refer to **DDS::SampleInfo** (p. 1148) and **DESTINATION_ORDER** (p. 292) QoS policy for details)

The constraints on the values of the `handle` parameter and the corresponding error behavior are the same specified for the **DDS::TypedDataWriter::write** (p. 1376) operation.

This operation may block and time out (**DDS::Retcode_Timeout** (p. 1124)) under the same circumtances described for **DDS::TypedDataWriter::write** (p. 1376).

If there are no instance resources left, this operation may fail with **DDS::Retcode_OutOfResources** (p. 1122). Calling **DDS::TypedDataWriter::unregister_instance** (p. 1372) may help free up some resources.

This operation may fail with **DDS::Retcode_BadParameter** (p. 1115) under the same circumstances described for the write operation.

Parameters:

instance_data *<<in>> (p. 175)* The data to write. This method will fail

with **DDS::Retcode_BadParameter** (p. 1115) if `instance_data` is NULL.

handle *<<in>>* *(p. 175)* Either the handle returned by a previous call to **DDS::TypedDataWriter::register_instance** (p. 1370), or else the special value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53). If **Foo** (p. 877) *has* a key and `handle` is not **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), `handle` must represent a registered instance of type **Foo** (p. 877). Otherwise, this method may fail with **DDS::Retcode_BadParameter** (p. 1115). This method will fail with **DDS::Retcode_BadParameter** (p. 1115) if `handle` is NULL.

source_timestamp *<<in>>* *(p. 175)* When using DDS::DestinationOrderQosPolicyKind::BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS the timestamp value must be greater than or equal to the timestamp value used in the last writer operation (*register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application-provided timestamp) However, if it is less than the timestamp of the previous operation but the difference is less than the **DDS::DestinationOrderQosPolicy::source_timestamp_tolerance** (p. 562), 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. 562), the function will return **DDS::Retcode_BadParameter** (p. 1115).

Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_Timeout** (p. 1124), **DDS::Retcode_OutOfResources** (p. 1122), or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::TypedDataWriter::write (p. 1376)
DDS::DataReader (p. 433)
DESTINATION_ORDER (p. 292)

6.224.2.7 template<typename T> void DDS::TypedDataWriter< T >::dispose (T *instance_data*, DDS::InstanceHandle_t% *instance_handle*) [inline]

Requests the middleware to delete the data.

This operation is useful only for keyed data types. Using it for non-keyed types has no effect and reports no error.

The actual deletion is postponed until there is no more use for that data in the whole system.

Applications are made aware of the deletion by means of operations on the **DDS::DataReader** (p. 433) objects that already knew that instance. **DDS::DataReader** (p. 433) objects that didn't know the instance will never see it.

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 Data Distribution Service will automatically supply the value of the `source_timestamp` that is made available to **DDS::DataReader** (p. 433) objects by means of the `source_timestamp` attribute inside the **DDS::SampleInfo** (p. 1148).

The constraints on the values of the handle parameter and the corresponding error behavior are the same specified for the **DDS::TypedDataWriter::unregister_instance** (p. 1372) operation.

The special value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) 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 `handle` is any value other than **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), then it must correspond to an instance that has been registered. If there is no correspondence, the operation will fail with **DDS::Retcode_BadParameter** (p. 1115).

RTI Data Distribution Service will not detect the error when the `handle` is any value other than **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), 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 Data Distribution Service will treat as if the **dispose()** (p. 1379) operation is for the instance as indicated by the `handle`.

This operation may block and time out (**DDS::Retcode_Timeout** (p. 1124)) under the same circumtances described for **DDS::TypedDataWriter::write()** (p. 1376).

If there are no instance resources left, this operation may fail with **DDS::Retcode_OutOfResources** (p. 1122). Calling **DDS::TypedDataWriter::unregister_instance** (p. 1372) may help freeing up some resources.

Parameters:

instance_data *<<in>>* *(p. 175)* The data to dispose. If **Foo**

(p. 877) *has* a key and `instance_handle` is **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used. If **Foo** (p. 877) *has* a key, `instance_data` can be NULL only if `instance_handle` is not **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53). Otherwise, this method will fail with **DDS::Retcode_BadParameter** (p. 1115).

instance_handle *<<in>> (p. 175)* Either the handle returned by a previous call to **DDS::TypedDataWriter::register_instance** (p. 1370), or else the special value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53). If **Foo** (p. 877) *has* a key and `instance_handle` is **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), `instance_handle` is not used and `instance` is deduced from `instance_data`. If **Foo** (p. 877) has no key, `instance_handle` is not used. If `handle` is used, it must represent a registered instance of type **Foo** (p. 877). Otherwise, this method fail with **DDS::Retcode_BadParameter** (p. 1115). This method will fail with **DDS::Retcode_BadParameter** (p. 1115) if `handle` is NULL. If **Foo** (p. 877) *has* a key, `instance_handle` cannot be **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) if `instance_data` is NULL. Otherwise, this method will fail with **DDS::Retcode_BadParameter** (p. 1115).

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_Timeout** (p. 1124), **DDS::Retcode_OutOfResources** (p. 1122) or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::TypedDataWriter::dispose_w_timestamp (p. 1381)
RELATIONSHIP BETWEEN REGISTRATION, LIVELINESS and OWNERSHIP (p. 995)

6.224.2.8 template<typename T> void DDS::TypedDataWriter< T >::dispose_w_timestamp (T *instance_data*, DDS::InstanceHandle_t% *instance_handle*, DDS::Time_t% *source_timestamp*) [inline]

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. 433) objects by means of the `source_timestamp` attribute inside the **DDS::SampleInfo** (p. 1148).

The constraints on the values of the `handle` parameter and the corresponding error behavior are the same specified for the **DDS::TypedDataWriter::dispose** (p. 1379) operation.

This operation may block and time out (**DDS::Retcode_Timeout** (p. 1124)) under the same circumtances described for **DDS::TypedDataWriter::write** (p. 1376).

If there are no instance resources left, this operation may fail with **DDS::Retcode_OutOfResources** (p. 1122). Calling **DDS::TypedDataWriter::unregister_instance** (p. 1372) may help freeing up some resources.

Parameters:

instance_data *<<in>>* *(p. 175)* The data to dispose. If **Foo** (p. 877) *has* a key and `instance_handle` is **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), only the fields that represent the key are examined by the function. Otherwise, `instance_data` is not used. If **Foo** (p. 877) *has* a key, `instance_data` can be NULL only if `instance_handle` is not **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53). Otherwise, this method will fail with **DDS::Retcode_BadParameter** (p. 1115).

instance_handle *<<in>>* *(p. 175)* Either the handle returned by a previous call to **DDS::TypedDataWriter::register_instance** (p. 1370), or else the special value **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53). If **Foo** (p. 877) *has* a key and `instance_handle` is **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), `instance_handle` is not used and `instance` is deduced from `instance_data`. If **Foo** (p. 877) has no key, `instance_handle` is not used. If `handle` is used, it must represent a registered instance of type **Foo** (p. 877). Otherwise, this method may fail with **DDS::Retcode_BadParameter** (p. 1115) This method will fail with **DDS::Retcode_BadParameter** (p. 1115) if `handle` is NULL. If **Foo** (p. 877) *has* a key, `instance_handle` cannot be **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53) if `instance_data` is NULL. Otherwise, this method will fail with **DDS::Retcode_BadParameter** (p. 1115).

source_timestamp *<<in>>* *(p. 175)* The timestamp value must be greater than or equal to the timestamp value used in the last writer operation (used in a *register*, *unregister*, *dispose*, or *write*, with either the automatically supplied timestamp or the application provided timestamp). This timestamp may potentially affect the order in which readers observe events from multiple writers. This timestamp will be available to the **DDS::DataReader** (p. 433) objects by means of the `source_timestamp` attribute inside the **DDS::SampleInfo** (p. 1148). Cannot be NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235), **DDS::Retcode_-Timeout** (p. 1124), **DDS::Retcode_OutOfResources** (p. 1122) or **DDS::Retcode_NotEnabled** (p. 1121).

See also:

DDS::TypedDataWriter::dispose (p. 1379)

6.224.2.9 template<typename T> void DDS::TypedDataWriter< T >::get_key_value (T *key_holder*, DDS::InstanceHandle_t% *handle*) [inline]

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 **Foo** (p. 877) has no key, this method has no effect and exit with no error.

For keyed data types, this operation may fail with **DDS::Retcode_-BadParameter** (p. 1115) if the `handle` does not correspond to an existing data-object known to the **DDS::DataWriter** (p. 499).

Parameters:

key_holder *<<inout>>* *(p. 176)* a user data type specific key holder, whose `key` fields are filled by this operation. If **Foo** (p. 877) has no key, this method has no effect. This method will fail with **DDS::Retcode_BadParameter** (p. 1115) if `key_holder` is NULL.

handle *<<in>>* *(p. 175)* the `instance` whose key is to be retrieved. If **Foo** (p. 877) *has* a key, `handle` must represent a registered instance of type **Foo** (p. 877). Otherwise, this method will fail with **DDS::Retcode_BadParameter** (p. 1115). If **Foo** (p. 877) *has* a key and `handle` is **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53), this method will fail with **DDS::Retcode_BadParameter** (p. 1115). This method will fail with **DDS::Retcode_BadParameter** (p. 1115) if `handle` is NULL.

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_-NotEnabled** (p. 1121).

See also:

DDS::TypedDataReader::get_key_value (p. 1365)

6.224.2.10 template<typename T> InstanceHandle_t DDS::TypedDataWriter< T >::lookup_instance (T *key_holder*) [inline]

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 Data Distribution Service is unable to provide an instance handle, RTI Data Distribution Service will return the special value HANDLE_NIL.

Parameters:

key_holder *<<in>>* *(*p. 175*)* a user data type specific key holder.

Returns:

the instance handle associated with this instance. If **Foo** (p. 877) has no key, this method has no effect and returns **DDS::InstanceHandle_t::HANDLE_NIL** (p. 53)

6.225 DDS::TypeSupport Class Reference

<<***interface***>> *(p. 175)* An abstract *marker* interface that has to be specialized for each concrete user data type that will be used by the application.

`#include <managed_topic.h>`

Inheritance diagram for DDS::TypeSupport::

6.225.1 Detailed Description

<<***interface***>> *(p. 175)* 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. 884), from a description of the type in IDL.

A **DDS::TypeSupport** (p. 1385) must be registered using the **FooTypeSupport::register_type** (p. 885) operation on this type-specific class before it can be used to create **DDS::Topic** (p. 1258) objects.

See also:

FooTypeSupport (p. 884)
rtiddsgen (p. 196)

6.226 DDS::TypeSupportQosPolicy Struct Reference

Allows you to attach application-specific values to a **DataWriter** (p. 499) or **DataReader** (p. 433) that are passed to the serialization or deserialization routine of the associated data type.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_typesupport_qos_policy_name** ()

  *Stringified human-readable name for **DDS::TypeSupportQosPolicy** (p. 1386).*

Public Attributes

- IntPtr **plugin_data**

  Value to pass into the type plugin's de-/serialization function.

6.226.1 Detailed Description

Allows you to attach application-specific values to a **DataWriter** (p. 499) or **DataReader** (p. 433) that are passed to the serialization or deserialization routine of the associated data type.

The purpose of this QoS is to allow a user application to pass data to a type plugin's support functions.

Entity:

DDS::DataReader (p. 433), **DDS::DataWriter** (p. 499)

Properties:

RxO (p. 268) = N/A

Changeable (p. 269) = **YES** (p. 269)

6.226.2 Usage

This QoS policy allows you to associate a pointer to an object with a **DDS::DataWriter** (p. 499) or **DDS::DataReader** (p. 433). This object pointer is passed to the serialization routine of the data type associated with the **DDS::DataWriter** (p. 499) or the deserialization routine of the data type associated with the **DDS::DataReader** (p. 433).

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. 499) or received by a **DDS::DataReader** (p. 433) is serialized or deserialized on a per **DataWriter** (p. 499) and **DataReader** (p. 433) 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** (p. 1258)) don't change, they only need to be sent once. The information passed in through the **TypeSupport** (p. 1385) QoS policy could be used to signal the serialization routine to send the row/column names the first time a **DDS::DataWriter** (p. 499) calls **DDS::TypedDataWriter::write** (p. 1376), and then never again.

6.226.3 Member Data Documentation

6.226.3.1 IntPtr DDS::TypeSupportQosPolicy::plugin_data

Value to pass into the type plugin's de-/serialization function.

[**default**] NULL

6.227 DDS::UDPv4Transport Interface Reference

Built-in transport plug-in using UDP/IPv4.

`#include <managed_transport.h>`

6.227.1 Detailed Description

Built-in transport plug-in using UDP/IPv4.

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.

The user can configure an instance of this plugin to only use unicast or only use multicast, see DDS::UDPv4Transport_Property_t::unicast_enabled and DDS::UDPv4Transport_Property_t::multicast_enabled.

In addition, the user 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 "white" and "black" lists in the base property's fields (DDS::Transport_Property_t::allow_interfaces_list, DDS::Transport_Property_t::deny_interfaces_list, DDS::Transport_Property_t::allow_multicast_interfaces_list, DDS::Transport_Property_t::deny_multicast_interfaces_list).

RTI Data Distribution Service can implicitly create this plugin and register with the **DDS::DomainParticipant** (p. 577) if this transport is specified in **DDS::TransportBuiltinQosPolicy** (p. 1285).

To specify the properties of the builtin UDPv4 transport that is implicitly registered, you can either:

- call DDS::Transport_Support::set_builtin_transport_property or
- specify the predefined property names in **DDS::PropertyQosPolicy** (p. 1023) associated with the **DDS::DomainParticipant** (p. 577). (see **UDPv4 Transport Property Names in Property QoS Policy of Domain Participant** (p. 1389)).

Builtin transport plugin properties specified in **DDS::PropertyQosPolicy** (p. 1023) always overwrite the ones specified through DDS::Transport_Support::set_builtin_transport_property(). 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 Data Distribution Service. Any properties set after the builtin transport is registered will be ignored. See **Built-in Transport Plugins** (p. 122) for details on when a builtin transport is registered.

6.227.2 UDPv4 Transport Property Names in Property QoS Policy of Domain Participant

The following table lists the predefined property names that can be set in **DDS::PropertyQosPolicy** (p. 1023) of a **DDS::DomainParticipant** (p. 577) to configure the builtin UDPv4 transport plugin.

See also:

DDS::Transport_Support::set_builtin_transport_property()

Property Name	Description
dds.transport.UDPv4.builtin.parent.address_bit_count	See DDS::Transport_Property_t::address_bit_count
dds.transport.UDPv4.builtin.parent.properties_bitmap	See DDS::Transport_Property_t::properties_bitmap
dds.transport.UDPv4.builtin.parent.gather_send_buffer_count_max	See DDS::Transport_Property_t::gather_send_buffer_count_max
dds.transport.UDPv4.builtin.parent.message_size_max	See DDS::Transport_Property_t::message_size_max
dds.transport.UDPv4.builtin.parent.allow_interfaces	See DDS::Transport_Property_t::allow_interfaces_list and DDS::Transport_Property_t::allow_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example, 127.0.0.1,eth0
dds.transport.UDPv4.builtin.parent.deny_interfaces	See DDS::Transport_Property_t::deny_interfaces_list and DDS::Transport_Property_t::deny_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.UDPv4.builtin.parent.allow_multicast_interfaces	See DDS::Transport_Property_t::allow_multicast_interfaces_list and DDS::Transport_Property_t::allow_multicast_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.UDPv4.builtin.parent.deny_multicast_interfaces	See DDS::Transport_Property_t::deny_multicast_interfaces_list and DDS::Transport_Property_t::deny_multicast_interfaces_list_length. Interfaces should be specified as comma-separated strings, with each comma delimiting an interface. For example: 127.0.0.1,eth0
dds.transport.UDPv4.builtin.send_socket_buffer_size	See DDS::UDPv4Transport_Property_t::send_socket_buffer_size
dds.transport.UDPv4.builtin.recv_socket_buffer_size	See DDS::UDPv4Transport_Property_t::recv_socket_buffer_size
dds.transport.UDPv4.builtin.unicast_enabled	See DDS::UDPv4Transport_Property_t::unicast_enabled
dds.transport.UDPv4.builtin.multicast_enabled	See DDS::UDPv4Transport_Property_t::multicast_enabled

6.228 DDS::UDPv6Transport Interface Reference

Built-in transport plug-in using UDP/IPv6.

`#include <managed_transport.h>`

6.228.1 Detailed Description

Built-in transport plug-in using UDP/IPv6.

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.

The user can configure an instance of this plugin to only use unicast or only use multicast, see DDS::UDPv6Transport_Property_t::unicast_enabled and DDS::UDPv6Transport_Property_t::multicast_enabled.

In addition, the user 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 "white" and "black" lists in the base property's fields (DDS::Transport_Property_t::allow_interfaces_list, DDS::Transport_Property_t::deny_interfaces_list, DDS::Transport_Property_t::allow_multicast_interfaces_list, DDS::Transport_Property_t::deny_multicast_interfaces_list).

RTI Data Distribution Service can implicitly create this plugin and register it with the **DDS::DomainParticipant** (p. 577) if this transport is specified in the **DDS::TransportBuiltinQosPolicy** (p. 1285).

To specify the properties of the builtin UDPv6 transport that is implicitly registered, you can either:

- call DDS::Transport_Support::set_builtin_transport_property or
- specify the predefined property names in **DDS::PropertyQosPolicy** (p. 1023) associated with the **DDS::DomainParticipant** (p. 577). (see **UDPv6 Transport Property Names in Property QoS Policy of Domain Participant** (p. 1392)).

Builtin transport plugin properties specified in **DDS::PropertyQosPolicy** (p. 1023) always overwrite the ones specified through DDS::Transport_Support::set_builtin_transport_property(). 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 Data Distribution Service. Any properties that are set after the builtin transport is registered will be ignored. See **Built-in Transport Plugins** (p. 122) for details on when a builtin transport is registered.

6.228.2 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. 1023) of a **DDS::DomainParticipant** (p. 577) to configure the builtin UDPv6 transport plugin.

See also:

DDS::Transport_Support::set_builtin_transport_property()

Property Name	Description
dds.transport.UDPv6.builtin.parent.address_bit_count	See DDS::Transport_Property_t::address_bit_count
dds.transport.UDPv6.builtin.parent.properties_bitmap	See DDS::Transport_Property_t::properties_bitmap
dds.transport.UDPv6.builtin.parent.gather_send_buffer_count_max	See DDS::Transport_Property_t::gather_send_buffer_count_max
dds.transport.UDPv6.builtin.parent.message_size_max	See DDS::Transport_Property_t::message_size_max
dds.transport.UDPv6.builtin.parent.allow_interfaces	See DDS::Transport_Property_t::allow_interfaces_list and DDS::Transport_Property_t::allow_interfaces_list_length. 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 DDS::Transport_Property_t::deny_interfaces_list and DDS::Transport_Property_t::deny_interfaces_list_length. 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 DDS::Transport_Property_t::allow_multicast_interfaces_list and DDS::Transport_Property_t::allow_multicast_interfaces_list_length. 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_multicast_interfaces	See DDS::Transport_Property_t::deny_multicast_interfaces_list and DDS::Transport_Property_t::deny_multicast_interfaces_list_length. 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.send_socket_buffer_size	See DDS::UDPv6Transport_Property_t::send_socket_buffer_size
dds.transport.UDPv6.builtin.recv_socket_buffer_size	See DDS::UDPv6Transport_Property_t::recv_socket_buffer_size
dds.transport.UDPv6.builtin.unicast_enabled	See DDS::UDPv6Transport_Property_t::unicast_enabled
dds.transport.UDPv6.builtin.multicast_	See DDS::UDPv6Transport_Property_t::multicast_enabled

6.229 DDS::UnionMember Class Reference

A description of a member of a union.

`#include <managed_typecode.h>`

Public Attributes

- ^ System::String^ **name**

  The name of the union member.

- ^ System::Boolean **is_pointer**

  Indicates whether the union member is a pointer or not.

- ^ **IntSeq**^ **labels**

  The labels of the union member.

- ^ **TypeCode**^ **type**

  The type of the union member.

6.229.1 Detailed Description

A description of a member of a union.

See also:

DDS::UnionMemberSeq (p. 1396)
DDS::TypeCodeFactory::create_union_tc (p. 1332)

6.229.2 Member Data Documentation

6.229.2.1 System::String ^ DDS::UnionMember::name

The name of the union member.

Cannot be null.

6.229.2.2 System::Boolean DDS::UnionMember::is_pointer

Indicates whether the union member is a pointer or not.

6.229.2.3 IntSeq ^ 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 System::Int32 the label value should be evaluated to an integer value. For instance, 'a' would be evaluated to 97.

6.229.2.4 TypeCode ^ DDS::UnionMember::type

The type of the union member.

Cannot be null.

6.230 DDS::UnionMemberSeq Class Reference

Defines a sequence of union members.

`#include <managed_typecode.h>`

Inheritance diagram for DDS::UnionMemberSeq::

6.230.1 Detailed Description

Defines a sequence of union members.

See also:

DDS::UnionMember (p. 1394)
DDS::Sequence (p. 1163)
DDS::TypeCodeFactory::create_union_tc (p. 1332)

6.231 DDS::UnsignedIntSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < System::UInt32 >.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::UnsignedIntSeq::

Public Member Functions

- ^ **UnsignedIntSeq** ()

  Constructs an empty sequence of unsigned integers with an initial maximum of zero.

- ^ **UnsignedIntSeq** (System::Int32 max)

  Constructs an empty sequence of unsigned integers with the given initial maximum.

- ^ **UnsignedIntSeq** (**UnsignedIntSeq**^ ints)

  Constructs a new sequence containing the given unsigned integers.

6.231.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < System::UInt32 >.

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

System::UInt32
DDS::Sequence (p. 1163)

6.231.2 Constructor & Destructor Documentation

6.231.2.1 DDS::UnsignedIntSeq::UnsignedIntSeq () `[inline]`

Constructs an empty sequence of unsigned integers with an initial maximum of zero.

6.231.2.2 DDS::UnsignedIntSeq::UnsignedIntSeq (System::Int32 *max*) [inline]

Constructs an empty sequence of unsigned integers with the given initial maximum.

6.231.2.3 DDS::UnsignedIntSeq::UnsignedIntSeq (UnsignedIntSeq^ *ints*) [inline]

Constructs a new sequence containing the given unsigned integers.

Parameters:

ints the initial contents of this sequence

6.232 DDS::UnsignedLongSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < System::UInt64 >.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::UnsignedLongSeq::

Public Member Functions

- ^ **UnsignedLongSeq** ()

  Constructs an empty sequence of unsigned long integers with an initial maximum of zero.

- ^ **UnsignedLongSeq** (System::Int32 max)

  Constructs an empty sequence of unsigned long integers with the given initial maximum.

- ^ **UnsignedLongSeq** (**UnsignedLongSeq**^ longs)

  Constructs a new sequence containing the given unsigned long integers.

6.232.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < System::UInt64 >.

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

System::UInt64
DDS::Sequence (p. 1163)

6.232.2 Constructor & Destructor Documentation

6.232.2.1 DDS::UnsignedLongSeq::UnsignedLongSeq () `[inline]`

Constructs an empty sequence of unsigned long integers with an initial maximum of zero.

6.232.2.2 DDS::UnsignedLongSeq::UnsignedLongSeq (System::Int32 *max*) [inline]

Constructs an empty sequence of unsigned long integers with the given initial maximum.

6.232.2.3 DDS::UnsignedLongSeq::UnsignedLongSeq (UnsignedLongSeq^ *longs*) [inline]

Constructs a new sequence containing the given unsigned long integers.

Parameters:

longs the initial contents of this sequence

6.233 DDS::UnsignedShortSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < System::UInt16 >.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::UnsignedShortSeq::

Public Member Functions

- ^ **UnsignedShortSeq** ()

  Constructs an empty sequence of unsigned short integers with an initial maximum of zero.

- ^ **UnsignedShortSeq** (System::Int32 max)

  Constructs an empty sequence of unsigned short integers with the given initial maximum.

- ^ **UnsignedShortSeq** (**UnsignedShortSeq**^ shorts)

  Constructs a new sequence containing the given unsigned short integers.

6.233.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < System::UInt16 >.

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

System::UInt16
DDS::Sequence (p. 1163)

6.233.2 Constructor & Destructor Documentation

6.233.2.1 DDS::UnsignedShortSeq::UnsignedShortSeq () `[inline]`

Constructs an empty sequence of unsigned short integers with an initial maximum of zero.

6.233.2.2 DDS::UnsignedShortSeq::UnsignedShortSeq (System::Int32 *max*) [inline]

Constructs an empty sequence of unsigned short integers with the given initial maximum.

6.233.2.3 DDS::UnsignedShortSeq::UnsignedShortSeq (UnsignedShortSeq^ *shorts*) [inline]

Constructs a new sequence containing the given unsigned short integers.

Parameters:

shorts the initial contents of this sequence

6.234 DDS::UserDataQosPolicy Class Reference

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 42) during discovery.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- ^ static System::String^ **get_userdata_qos_policy_name** ()

  *Stringified human-readable name for **DDS::UserDataQosPolicy** (p. 1403).*

Public Attributes

- ^ **ByteSeq^ value**

  a sequence of octets

6.234.1 Detailed Description

Attaches a buffer of opaque data that is distributed by means of **Built-in Topics** (p. 42) during discovery.

Entity:

DDS::DomainParticipant (p. 577), **DDS::DataReader** (p. 433), **DDS::DataWriter** (p. 499)

Properties:

RxO (p. 268) = NO;
Changeable (p. 269) = **YES** (p. 269)

See also:

DDS::DomainParticipant::get_builtin_subscriber (p. 632)

6.234.2 Usage

The purpose of this QoS is to allow the application to attach additional information to the created **DDS::Entity** (p. 845) 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 Data Distribution Service.

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. 633), **DDS::DomainParticipant::ignore_publication** (p. 635), **DDS::DomainParticipant::ignore_subscription** (p. 636), and **DDS::DomainParticipant::ignore_topic** (p. 634), 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 Data Distribution Service stores the data placed in this policy in pre-allocated pools. It is therefore necessary to configure RTI Data Distribution Service 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. 701), **DDS::DomainParticipantResourceLimitsQosPolicy::writer_user_data_max_length** (p. 702), and **DDS::DomainParticipantResourceLimitsQosPolicy::reade user_data_max_length** (p. 702).

6.234.3 Member Data Documentation

6.234.3.1 ByteSeq ^ DDS::UserDataQosPolicy::value

a sequence of octets

[default] empty (zero-length)

[range] Octet sequence of length [0,max_length]

6.235 DDS::ValueMember Class Reference

A description of a member of a value type.

`#include <managed_typecode.h>`

Public Attributes

- System::String^ **name**

  The name of the value member.

- **TypeCode^ type**

  The type of the value member.

- System::Boolean **is_pointer**

  Indicates whether the value member is a pointer or not.

- System::Int16 **bits**

  Number of bits of a bitfield member.

- System::Boolean **is_key**

  Indicates if the value member is a key member or not.

- **Visibility access**

  The type of access (public, private) for the value member.

6.235.1 Detailed Description

A description of a member of a value type.

See also:

DDS::ValueMemberSeq (p. 1407)
DDS::TypeCodeFactory::create_value_tc (p. 1332)

6.235.2 Member Data Documentation

6.235.2.1 System::String ^ DDS::ValueMember::name

The name of the value member.

Cannot be null.

6.235.2.2 TypeCode ^ DDS::ValueMember::type

The type of the value member.

Cannot be null.

6.235.2.3 System::Boolean DDS::ValueMember::is_pointer

Indicates whether the value member is a pointer or not.

6.235.2.4 System::Int16 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. 65).

6.235.2.5 System::Boolean DDS::ValueMember::is_key

Indicates if the value member is a key member or not.

6.235.2.6 Visibility DDS::ValueMember::access

The type of access (public, private) for the value member.

It can take the values: DDS::Visibility::PRIVATE_MEMBER or DDS::Visibility::PUBLIC_MEMBER.

6.236 DDS::ValueMemberSeq Class Reference

Defines a sequence of value members.

`#include <managed_typecode.h>`

Inheritance diagram for DDS::ValueMemberSeq::

6.236.1 Detailed Description

Defines a sequence of value members.

See also:

DDS::ValueMember (p. 1405)
DDS::Sequence (p. 1163)
DDS::TypeCodeFactory::create_value_tc (p. 1332)

6.237 DDS::VendorId_t Struct Reference

<<*eXtension*>> *(p. 174)* Type used to represent the vendor of the service implementing the RTPS protocol.

`#include <managed_infrastructure.h>`

Public Attributes

- System::Byte **vendor_id_1**

  The vendor Id.

- System::Byte **vendor_id_2**

  The vendor Id.

Properties

- static **VendorId_t VENDORID_UNKNOWN** `[get]`

  The ID used when the vendor of the service implementing the RTPS protocol is not known.

- static System::Int32 **VENDORID_LENGTH_MAX** `[get]`

  Length of vendor id.

6.237.1 Detailed Description

<<*eXtension*>> *(p. 174)* Type used to represent the vendor of the service implementing the RTPS protocol.

6.237.2 Member Data Documentation

6.237.2.1 System::Byte DDS::VendorId_t::vendor_id_1

The vendor Id.

6.237.2.2 System::Byte DDS::VendorId_t::vendor_id_2

The vendor Id.

6.238 DDS::ViewStateKind Struct Reference

Indicates whether or not an instance is new.

`#include <managed_subscription.h>`

Properties

- static **ViewStateKind NEW_VIEW_STATE** `[get]`

  New instance. This latest generation of the instance has not previously been accessed.

- static **ViewStateKind NOT_NEW_VIEW_STATE** `[get]`

  Not a new instance. This latest generation of the instance has previously been accessed.

- static **ViewStateKind ANY_VIEW_STATE** `[get]`

  Any view state ***DDS::ViewStateKind::NEW_VIEW_STATE*** *(p. 1410)* | ***DDS::ViewStateKind::NOT_NEW_VIEW_STATE*** *(p. 1410).*

6.238.1 Detailed Description

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. 433). The view state can be either:

- **DDS::ViewStateKind::NEW_VIEW_STATE** (p. 1410) indicates that either this is the first time that the **DDS::DataReader** (p. 433) has ever accessed samples of that instance, or else that the **DDS::DataReader** (p. 433) 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. 1153) and the **DDS::SampleInfo::no_writers_generation_count** (p. 1154).

- **DDS::ViewStateKind::NOT_NEW_VIEW_STATE** (p. 1410) indicates that the **DDS::DataReader** (p. 433) 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. 1148) is a snapshot of the view state of the instance relative to the **DDS::DataReader** (p. 433) 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. 433), the middleware can reclaim all local resources regarding the instance. Future samples will be treated as "never seen."

6.238.2 Property Documentation

6.238.2.1 ViewStateKind DDS::ViewStateKind::NEW_VIEW_STATE [static, get]

New instance. This latest generation of the instance has not previously been accessed.

6.238.2.2 ViewStateKind DDS::ViewStateKind::NOT_NEW_VIEW_STATE [static, get]

Not a new instance. This latest generation of the instance has previously been accessed.

6.239 DDS::WaitSet Class Reference

<<***interface***>> *(p. 175)* Allows an application to wait until one or more of the attached **DDS::Condition** (p. 408) objects has a `trigger_value` of true or else until the timeout expires.

`#include <managed_infrastructure.h>`

Public Member Functions

- ^ void **wait** (**ConditionSeq**^ active_conditions, **Duration_t** timeout)

  Allows an application thread to wait for the occurrence of certain conditions.

- ^ void **attach_condition** (**Condition**^ cond)

  *Attaches a **DDS::Condition** (p. 408) to the **DDS::WaitSet** (p. 1411).*

- ^ void **detach_condition** (**Condition**^ cond)

  *Detaches a **DDS::Condition** (p. 408) from the **DDS::WaitSet** (p. 1411).*

- ^ void **get_conditions** (**ConditionSeq**^ attached_conditions)

  *Retrieves the list of attached **DDS::Condition** (p. 408) (s).*

- ^ void **set_property** (**WaitSetProperty_t** prop)

  <<**eXtension**>> (p. 174) *Sets the **DDS::WaitSetProperty_t** (p. 1419), to configure the associated **DDS::WaitSet** (p. 1411) to return after one or more trigger events have occurred.*

- ^ void **get_property** (**WaitSetProperty_t**% prop)

  <<**eXtension**>> (p. 174) *Retrieves the **DDS::WaitSetProperty_t** (p. 1419) configuration of the associated **DDS::WaitSet** (p. 1411).*

- ^ virtual ~**WaitSet** ()

  Destructor.

- ^ **WaitSet** ()

  Default no-argument constructor.

- ^ **WaitSet** (**WaitSetProperty_t**% prop)

  <<**eXtension**>> (p. 174) *Constructor for a **DDS::WaitSet** (p. 1411) that may delay for more while specifying that will be woken up after the given number of events or delay period, whichever happens first*

6.239.1 Detailed Description

<<*interface*>> *(p. 175)* Allows an application to wait until one or more of the attached **DDS::Condition** (p. 408) objects has a `trigger_value` of true or else until the timeout expires.

6.239.2 Usage

DDS::Condition (p. 408) (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. 408) objects (**DDS::StatusCondition** (p. 1183), **DDS::ReadCondition** (p. 1084) or **DDS::QueryCondition** (p. 1082)) and attaching them to a **DDS::WaitSet** (p. 1411).

- It then waits on that **DDS::WaitSet** (p. 1411) until the `trigger_value` of one or several **DDS::Condition** (p. 408) objects become true.

- It then uses the result of the wait (i.e., `active_conditions`, the list of **DDS::Condition** (p. 408) objects with `trigger_value` == true) to actually get the information:

  - by calling **DDS::Entity::get_status_changes** (p. 850) and then `get_<communication_status>()` on the relevant **DDS::Entity** (p. 845), if the condition is a **DDS::StatusCondition** (p. 1183) and the status changes, refer to plain communication status;
  - by calling **DDS::Entity::get_status_changes** (p. 850) and then **DDS::Subscriber::get_datareaders** (p. 1218) on the relevant **DDS::Subscriber** (p. 1201) (and then **DDS::TypedDataReader::read()** (p. 1341) or **DDS::TypedDataReader::take** (p. 1342) on the returned **DDS::DataReader** (p. 433) objects), if the condition is a **DDS::StatusCondition** (p. 1183) and the status changes refers to DDS::StatusKind::DATA_ON_READERS_STATUS;
  - by calling **DDS::Entity::get_status_changes** (p. 850) and then **DDS::TypedDataReader::read()** (p. 1341) or **DDS::TypedDataReader::take** (p. 1342) on the relevant **DDS::DataReader** (p. 433), if the condition is a **DDS::StatusCondition** (p. 1183) and the status changes refers to DDS::StatusKind::DATA_AVAILABLE_STATUS;

- by calling directly **DDS::TypedDataReader::read_w_condition** (p. 1349) or **DDS::TypedDataReader::take_w_condition** (p. 1350) on a **DDS::DataReader** (p. 433) with the **DDS::Condition** (p. 408) as a parameter if it is a **DDS::ReadCondition** (p. 1084) or a **DDS::QueryCondition** (p. 1082).

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. 408) objects), **DDS::Condition** (p. 408) objects are meant to embed all that is needed to react properly when enabled. In particular, **DDS::Entity** (p. 845)-related conditions are related to exactly one **DDS::Entity** (p. 845) and cannot be shared.

The blocking behavior of the **DDS::WaitSet** (p. 1411) is illustrated below.

The result of a **DDS::WaitSet::wait** (p. 1416) operation depends on the state of the **DDS::WaitSet** (p. 1411), which in turn depends on whether at least one attached **DDS::Condition** (p. 408) has a `trigger_value` of true. If the wait operation is called on **DDS::WaitSet** (p. 1411) with state BLOCKED, it will block the calling thread. If wait is called on a **DDS::WaitSet** (p. 1411) with state UNBLOCKED, it will return immediately. In addition, when the **DDS::WaitSet** (p. 1411) transitions from BLOCKED to UNBLOCKED it wakes up any threads that had called wait on it.

A key aspect of the **DDS::Condition** (p. 408)/**DDS::WaitSet** (p. 1411) mechanism is the setting of the `trigger_value` of each **DDS::Condition** (p. 408).

6.239.3 Trigger State of a ::DDS::StatusCondition

The `trigger_value` of a **DDS::StatusCondition** (p. 1183) is the boolean OR of the ChangedStatusFlag of all the communication statuses (see **Status Kinds** (p. 238)) to which it is sensitive. That is, `trigger_value` == false only if all the values of the ChangedStatusFlags are false.

The sensitivity of the **DDS::StatusCondition** (p. 1183) 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. 1184) operation.

6.239.4 Trigger State of a ::DDS::ReadCondition

Similar to the **DDS::StatusCondition** (p. 1183), a **DDS::ReadCondition** (p. 1084) also has a `trigger_value` that determines whether the at-

tached **DDS::WaitSet** (p. 1411) is BLOCKED or UNBLOCKED. However, unlike the **DDS::StatusCondition** (p. 1183), the `trigger_value` of the **DDS::ReadCondition** (p. 1084) is tied to the presence of *at least a sample* managed by RTI Data Distribution Service with **DDS::SampleStateKind** (p. 1161) and **DDS::ViewStateKind** (p. 1409) matching those of the **DDS::ReadCondition** (p. 1084). Furthermore, for the **DDS::QueryCondition** (p. 1082) to have a `trigger_value` == true, the data associated with the sample must be such that the `query_expression` evaluates to true.

The fact that the trigger_value of a **DDS::ReadCondition** (p. 1084) depends on the presence of samples on the associated **DDS::DataReader** (p. 433) implies that a single take operation can potentially change the `trigger_value` of several **DDS::ReadCondition** (p. 1084) or **DDS::QueryCondition** (p. 1082) conditions. For example, if all samples are taken, any **DDS::ReadCondition** (p. 1084) and **DDS::QueryCondition** (p. 1082) conditions associated with the **DDS::DataReader** (p. 433) 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. 1411) 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. 1411), the condition transitioning to `trigger_value==FALSE` does not necessarily 'unwakeup' the **WaitSet** (p. 1411) as 'unwakening' may not be possible in general.

The consequence is that an application blocked on a **DDS::WaitSet** (p. 1411) 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. 1411) objects and taking data associated with the same **DDS::DataReader** (p. 433) entity.

To elaborate further, consider the following example: A **DDS::ReadCondition** (p. 1084) that has a `sample_state_mask` = {**DDS::SampleStateKind::NOT_READ_SAMPLE_STATE** (p. 1162)} will have `trigger_value` of true whenever a new sample arrives and will transition to false 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 Data Distribution Service). However if the same **DDS::ReadCondition** (p. 1084) had a `sample_state_mask` = { **DDS::SampleStateKind::READ_SAMPLE_STATE** (p. 1161), **DDS::SampleStateKind::NOT_READ_SAMPLE_STATE** (p. 1162) }, then the `trigger_value` would only become false 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. 1084).

6.239.5 Trigger State of a ::DDS::GuardCondition

The `trigger_value` of a **DDS::GuardCondition** (p. 892) is completely controlled by the application via the operation **DDS::GuardCondition::set_trigger_value** (p. 893).

See also:

Status Kinds (p. 238)
DDS::StatusCondition (p. 1183), **DDS::GuardCondition** (p. 892)
DDS::Listener (p. 952)

6.239.6 Constructor & Destructor Documentation

6.239.6.1 virtual DDS::WaitSet::~WaitSet () [virtual]

Destructor.

Releases the resources asociated with this **DDS::WaitSet** (p. 1411).

Calling this method multiple times on the same **DDS::WaitSet** (p. 1411) is safe; subsequent deletions will have no effect.

6.239.6.2 DDS::WaitSet::WaitSet ()

Default no-argument constructor.

Construct a new **DDS::WaitSet** (p. 1411).

Returns:

A new **DDS::WaitSet** (p. 1411) or null if one could not be allocated.

6.239.6.3 DDS::WaitSet::WaitSet (WaitSetProperty_t% *prop*)

<<eXtension>> *(p. 174)* Constructor for a **DDS::WaitSet** (p. 1411) 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. 1411).

Returns:

A new **DDS::WaitSet** (p. 1411) or null if one could not be allocated.

6.239.7 Member Function Documentation

6.239.7.1 void DDS::WaitSet::wait (ConditionSeq^ *active_conditions*, 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. 1411) have a `trigger_value` of true, 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 true (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. 408) objects is true, wait will return with the return code **DDS::Retcode_Timeout** (p. 1124). In this case, the resulting list of conditions will be empty.

DDS::Retcode_Timeout (p. 1124) will *not* be returned when the `timeout` duration is exceeded if attached **DDS::Condition** (p. 408) objects are true, or in the case of a **DDS::WaitSet** (p. 1411) waiting for more than one trigger event, if one or more trigger events have occurred.

It is not allowable for for more than one application thread to be waiting on the same **DDS::WaitSet** (p. 1411). If the wait operation is invoked on a **DDS::WaitSet** (p. 1411) that already has a thread blocking on it, the operation will return immediately with the value **DDS::Retcode_PreconditionNotMet** (p. 1123).

Parameters:

active_conditions *<<inout>>* *(p. 176)* a valid non-null **DDS::ConditionSeq** (p. 410) object. Note that RTI Data Distribution Service will not allocate a new object if `active_conditions` is null; the method will return **DDS::Retcode_PreconditionNotMet** (p. 1123).

timeout *<<in>>* *(p. 175)* a wait timeout

Exceptions:

One of the **Standard Return Codes** (p. 235) or **DDS::Retcode_PreconditionNotMet** (p. 1123) or **DDS::Retcode_Timeout** (p. 1124).

6.239.7.2 void DDS::WaitSet::attach_condition (Condition^ *cond*)

Attaches a **DDS::Condition** (p. 408) to the **DDS::WaitSet** (p. 1411).

It is possible to attach a **DDS::Condition** (p. 408) on a **DDS::WaitSet** (p. 1411) that is currently being waited upon (via the wait operation). In this case, if the **DDS::Condition** (p. 408) has a `trigger_value` of true, then attaching the condition will unblock the **DDS::WaitSet** (p. 1411).

Parameters:

cond *<<in>> (p. 175)* Contition to be attached.

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_OutOfResources** (p. 1122).

6.239.7.3 void DDS::WaitSet::detach_condition (Condition^ *cond*)

Detaches a **DDS::Condition** (p. 408) from the **DDS::WaitSet** (p. 1411).

If the **DDS::Condition** (p. 408) was not attached to the **DDS::WaitSet** (p. 1411) the operation will return **DDS::Retcode_BadParameter** (p. 1115).

Parameters:

cond *<<in>> (p. 175)* **Condition** (p. 408) to be detached.

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_PreconditionNotMet** (p. 1123).

6.239.7.4 void DDS::WaitSet::get_conditions (ConditionSeq^ *attached_conditions*)

Retrieves the list of attached **DDS::Condition** (p. 408) (s).

Parameters:

attached_conditions *<<inout>> (p. 176)* a **DDS::ConditionSeq** (p. 410) object where the list of attached conditions will be returned

Exceptions:

One of the **Standard Return Codes** (p. 235), or **DDS::Retcode_PreconditionNotMet** (p. 1123).

6.239.7.5 void DDS::WaitSet::set_property (WaitSetProperty_t *prop*)

*<<**eXtension**>> (p. 174)* Sets the **DDS::WaitSetProperty_t** (p. 1419), to configure the associated **DDS::WaitSet** (p. 1411) to return after one or more trigger events have occurred.

Parameters:

prop *<<**in**>> (p. 175)*

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.239.7.6 void DDS::WaitSet::get_property (WaitSetProperty_t% *prop*)

*<<**eXtension**>> (p. 174)* Retrieves the **DDS::WaitSetProperty_t** (p. 1419) configuration of the associated **DDS::WaitSet** (p. 1411).

Parameters:

prop *<<**out**>> (p. 176)*

Exceptions:

One of the **Standard Return Codes** (p. 235)

6.240 DDS::WaitSetProperty_t Struct Reference

<<*eXtension*>> *(p. 174)* Specifies the **DDS::WaitSet** (p. 1411) behavior for multiple trigger events.

`#include <managed_infrastructure.h>`

Public Attributes

- System::Int32 **max_event_count**

  Maximum number of trigger events to cause a ***DDS::WaitSet*** *(p. 1411) to awaken.*

- **Duration_t max_event_delay**

  Maximum delay from occurrence of first trigger event to cause a ***DDS::WaitSet*** *(p. 1411) to awaken.*

6.240.1 Detailed Description

<<*eXtension*>> *(p. 174)* Specifies the **DDS::WaitSet** (p. 1411) behavior for multiple trigger events.

In simple use, a **DDS::WaitSet** (p. 1411) returns when a single trigger event occurs on one of its attached **DDS::Condition** (p. 408) (s), or when the `timeout` maximum wait duration specified in the **DDS::WaitSet::wait** (p. 1416) call expires.

The **DDS::WaitSetProperty_t** (p. 1419) allows configuration of a **DDS::WaitSet** (p. 1411) to wait for up to `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. 1416) call continues to apply.

Entity:

DDS::WaitSet (p. 1411)

Properties:

RxO (p. 268) = N/A

Changeable (p. 269) = **YES** (p. 269)

6.240.2 Member Data Documentation

6.240.2.1 System::Int32 DDS::WaitSetProperty_t::max_event_count

Maximum number of trigger events to cause a **DDS::WaitSet** (p. 1411) to awaken.

The **DDS::WaitSet** (p. 1411) will wait until up to `max_event_count` trigger events have occurred before returning. The **DDS::WaitSet** (p. 1411) 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

;

6.240.2.2 Duration_t DDS::WaitSetProperty_t::max_event_delay

Maximum delay from occurrence of first trigger event to cause a **DDS::WaitSet** (p. 1411) to awaken.

The **DDS::WaitSet** (p. 1411) 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. 1411).

Note that **DDS::Retcode_Timeout** (p. 1124) is *not* returned if `max_event_delay` is exceeded. **DDS::Retcode_Timeout** (p. 1124) is returned only if the `timeout` duration expires before any trigger events occur.

[default] **DDS::Duration_t::DURATION_INFINITE** (p. 253);

6.241 DDS::WcharSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < System::Char >.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::WcharSeq::

Public Member Functions

- ^ **WcharSeq** ()

  Constructs an empty sequence of wide characters with an initial maximum of zero.

- ^ **WcharSeq** (System::Int32 max)

  Constructs an empty sequence of wide characters with the given initial maximum.

- ^ **WcharSeq** (**WcharSeq**^ chars)

  Constructs a new sequence containing the given wide characters.

6.241.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < System::Char >.

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

System::Char
DDS::Sequence (p. 1163)

6.241.2 Constructor & Destructor Documentation

6.241.2.1 DDS::WcharSeq::WcharSeq () `[inline]`

Constructs an empty sequence of wide characters with an initial maximum of zero.

6.241.2.2 DDS::WcharSeq::WcharSeq (System::Int32 *max*) [inline]

Constructs an empty sequence of wide characters with the given initial maximum.

6.241.2.3 DDS::WcharSeq::WcharSeq (WcharSeq^ *chars*) [inline]

Constructs a new sequence containing the given wide characters.

Parameters:

chars the initial contents of this sequence

6.242 DDS::WireProtocolQosPolicy Struct Reference

Specifies the wire-protocol-related attributes for the **DDS::DomainParticipant** (p. 577).

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_wireprotocol_qos_policy_name** ()

  *Stringified human-readable name for **DDS::WireProtocolQosPolicy** (p. 1423).*

Public Attributes

- System::Int32 **participant_id**

  A value used to distinguish between different participants belonging to the same domain on the same host.

- System::UInt32 **rtps_host_id**

  The RTPS Host ID of the domain participant.

- System::UInt32 **rtps_app_id**

  The RTPS App ID of the domain participant.

- System::UInt32 **rtps_instance_id**

  The RTPS Instance ID of the domain participant.

- **RtpsWellKnownPorts_t rtps_well_known_ports**

  Configures the RTPS well-known port mappings.

- System::Int32 **rtps_reserved_port_mask**

  Specifies which well-known ports to reserve when enabling the participant.

- **WireProtocolQosPolicyAutoKind rtps_auto_id_kind**

  Kind of auto mechanism used to calculate the GUID prefix.

Properties

^ static System::UInt32 **RTPS_AUTO_ID** [get]

Indicates that RTI Data Distribution Service should choose an appropriate host, app, instance or object ID automatically.

6.242.1 Detailed Description

Specifies the wire-protocol-related attributes for the **DDS::DomainParticipant** (p. 577).

Entity:

DDS::DomainParticipant (p. 577)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **NO** (p. 269)

6.242.2 Usage

This QoS policy configures some participant-wide properties of the DDS on-the-wire protocol, RTPS. (**DDS::DataWriterProtocolQosPolicy** (p. 529) and **DDS::DataReaderProtocolQosPolicy** (p. 465) configure RTPS and reliability properties on a per **DDS::DataWriter** (p. 499) or **DDS::DataReader** (p. 433) basis.)

NOTE: The default Qos policies returned by RTI Data Distribution Service contain the correctly initialized wire protocol attributes. The defaults are not expected to be modified normally, 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. 577) 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. 577) must have a unique value of this prefix relative to all other participants in its domain. In order to make it easier to control how this 96-bit value is generated, RTI Data Distribution Service divides it into three integers: a *host ID*, the value of which is based on the identity of the identity of the machine on which the participant is executing, an *application ID*, the value of which

is based on the process or task in which the particpant is contained, and an *instance ID*, which identifies the participant itself.

This QoS policy provides you with a choiche of algorithms for generating these values automatically. In case none of these algorithms suit your needs, you may also choose to specify some or all of them yourself.

The following three fields:

- **DDS::WireProtocolQosPolicy::rtps_host_id** (p. 1428)
- **DDS::WireProtocolQosPolicy::rtps_app_id** (p. 1429)
- **DDS::WireProtocolQosPolicy::rtps_instance_id** (p. 1429)

will compose the GUID prefix and by default are set to **DDS::WireProtocolQosPolicy::RTPS_AUTO_ID** (p. 330). The meaning of this flag depends on the value assigned to the **DDS::WireProtocolQosPolicy::rtps_auto_id_kind** (p. 1430) field.

Depending on the **DDS::WireProtocolQosPolicy::rtps_auto_id_kind** (p. 1430) value, there are two different scenarios:

1. In the default and most common scenario, **DDS::WireProtocolQosPolicy::rtps_auto_id_kind** (p. 1430) is set to DDS::WireProtocolQosPolicyAutoKind::RTPS_AUTO_ID_FROM_IP. Doing so, each field is interpeted 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
   - **rtps_app_id:** the process (or task) ID is assigned
   - **rtps_instance_id:** A counter is assigned that is incremented per new participant

   NOTE: If the IP assigned to the interface is not unique within the network (for instance, if it is not configured), then is it possible that the GUID (specifically, the rtps_host_id portion) may also not be unique.
2. In this situation, RTI Data Distribution Service provides a different value for rtps_auto_id_kind: DDS::WireProtocolQosPolicyAutoKind::RTPS_AUTO_ID_FROM_MAC. As the name suggests, this alternative mechanism will use the MAC address instead of the IPv4 address. Since the MAC address size is up to 64 bits, the logical mapping of the host information, the application ID, and the instance identifiers has to change.

   Note to Solaris Users: To use DDS::WireProtocolQosPolicyAutoKind::RTPS_AUTO_ID_FROM_MAC, you must run the RTI Data Distribution Service application while logged in as root.

   Using DDS::WireProtocolQosPolicyAutoKind::RTPS_AUTO_ID_FROM_MAC, the default value of each field is interpreted 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. In both scenarios, you can change the value of each field independently.

If DDS::WireProtocolQosPolicyAutoKind::RTPS_AUTO_ID_FROM_MAC is used, the rtps_instance_id has been logically split into two parts: 24 bits for the process/task ID and 8 bits for the per new participant counter. To give to users the ability to manually set the two parts indepently, a bit field mechanism has been introduced for the rtps_instance_id field when it is used in combination with DDS::WireProtocolQosPolicyAutoKind::RTPS_AUTO_ID_FROM_MAC. If one of the two parts is set to 0, only this part will be handled by RTI Data Distribution Service and you will be able to handle the other one manually.

Some examples are provided to better explain the behavior of this QoSPolicy in case you want to change the default behavior with DDS::WireProtocolQosPolicyAutoKind::RTPS_AUTO_ID_FROM_MAC.

First, get the participant QoS from the ParticipantFactory:

```
DomainParticipantFactory.TheParticipantFactory.
                get_default_participant_qos(participant_qos); <P>
```

Second, change the **DDS::WireProtocolQosPolicy** (p. 1423) using one of the options shown below.

Third, create the **DDS::DomainParticipant** (p. 577) as usual using the modified QoS structure instead of the default one.

Option 1: Use DDS::WireProtocolQosPolicyAutoKind::RTPS_AUTO_ID_FROM_MAC to explicitly set just the application/task identifier portion of the **rtps_instance_id** field.

```
participant_qos.wire_protocol.rtps_auto_id_kind =
                WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_MAC;
 participant_qos.wire_protocol.rtps_host_id      =
                WireProtocolQosPolicy.RTPS_AUTO_ID;
 participant_qos.wire_protocol.rtps_app_id       =
                WireProtocolQosPolicy.RTPS_AUTO_ID;
 participant_qos.wire_protocol.rtps_instance_id  = (/* App ID */ (12 << 8) |
                                                    /* Instance ID*/ (WireProtocolQosPolicy.RTPS_AUTO_ID))
```

Option 2: Handle only the per participant counter and let RTI Data Distribution Service handle the application/task identifier:

```
participant_qos.wire_protocol.rtps_auto_id_kind =
                        WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_MAC;
 participant_qos.wire_protocol.rtps_host_id      = WireProtocolQosPolicy.RTPS_AUTO_ID;
 participant_qos.wire_protocol.rtps_app_id       = WireProtocolQosPolicy.RTPS_AUTO_ID;
 participant_qos.wire_protocol.rtps_instance_id  =  (/* App ID */ (WireProtocolQosPolicy.RTPS_AUTO_ID) |
                                                    /* Instance ID*/ (12));
```

Option 3: Handle the entire **rtps_instance_id** field yourself:

```
participant_qos.wire_protocol.rtps_auto_id_kind = WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_MAC;
 participant_qos.wire_protocol.rtps_host_id      = WireProtocolQosPolicy.RTPS_AUTO_ID;
 participant_qos.wire_protocol.rtps_app_id       = WireProtocolQosPolicy.RTPS_AUTO_ID;
 participant_qos.wire_protocol.rtps_instance_id  = ( /* App ID */ (12 << 8)) |
                                                     /* Instance ID */ (9) )
```

NOTE: If you are using DDS::WireProtocolQosPolicyAutoKind::RTPS_AUTO_ID_FROM_MAC 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 Data Distribution Service will take responsibility for them). RTI recommends that you always specify the two parts separately in order to avoid errors.

Option 4: Let RTI Data Distribution Service handle the entire **rtps_instance_id** field:

```
participant_qos.wire_protocol.rtps_auto_id_kind = WireProtocolQosPolicyAutoKind.RTPS_AUTO_ID_FROM_MAC;
 participant_qos.wire_protocol.rtps_host_id      = WireProtocolQosPolicy.RTPS_AUTO_ID;
 participant_qos.wire_protocol.rtps_app_id       = WireProtocolQosPolicy.RTPS_AUTO_ID;
 participant_qos.wire_protocol.rtps_instance_id  = WireProtocolQosPolicy.RTPS_AUTO_ID;
```

NOTE: If you are using DDS::WireProtocolQosPolicyAutoKind::RTPS_AUTO_ID_FROM_MAC 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 Data Distribution Service will take responsibility for them). RTI recommends that you always specify the two parts separately in order to clearly show the difference.

6.242.3 Member Data Documentation

6.242.3.1 System::Int32 DDS::WireProtocolQosPolicy::participant_id

A value used to distinguish between 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. 484) or by the **DDS::DataWriterQos::unicast** (p. 550)).

For more information on port mapping, please refer to **DDS::RtpsWellKnownPorts_t** (p. 1142).

Each **DDS::DomainParticipant** (p. 577) in the same domain, 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 Data Distribution Service will *automatically* resolve the participant ID as follows.

- RTI Data Distribution Service will pick the *smallest* participant ID, based on the unicast ports available on the transports enabled for discovery.
- RTI Data Distribution Service will attempt to resolve an automatic port index either when a DomainPartcipant is enabled, or when a **DataReader** (p. 433) or a **DataWriter** (p. 499) is created. Therefore, all the transports enabled for discovery must have been registered by this time. Otherwise, the discovery transports registered after resolving the automatic port index may produce port conflicts when the DomainPartcipant is enabled.

[default] -1 [automatic], i.e. RTI Data Distribution Service will automatically pick the `participant_id`, as described above.

[range] [>= 0], or -1, and does not violate guidelines stated in **DDS::RtpsWellKnownPorts_t** (p. 1142).

See also:

DDS::Entity::enable() (p. 848)

6.242.3.2 System::UInt32 DDS::WireProtocolQosPolicy::rtps_host_id

The RTPS Host ID of the domain participant.

A machine/operating system specific host ID, that is unique in the domain.

[default] **DDS::WireProtocolQosPolicy::RTPS_AUTO_ID** (p. 330). The default value is interpreted as follows:

If **DDS::WireProtocolQosPolicy::rtps_auto_id_kind** (p. 1430) is equals to *RTPS_AUTO_ID_FROM_IP* (default value for this field) 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. 1430) is equals to *RTPS_AUTO_ID_FROM_MAC* the value will be interpreted as the first 32 bit of the MAC address assigned to the *first* up and running interface of the host machine.

[range] [0,0xffffffff]

6.242.3.3 System::UInt32 DDS::WireProtocolQosPolicy::rtps_app_id

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::WireProtocolQosPolicy::RTPS_AUTO_ID (p. 330). The default value is interpreted as follows:

If **DDS::WireProtocolQosPolicy::rtps_auto_id_kind** (p. 1430) is equals to *RTPS_AUTO_ID_FROM_IP* (default value for this field) the value will be the process (or task) ID.

If **DDS::WireProtocolQosPolicy::rtps_auto_id_kind** (p. 1430) is equals to *RTPS_AUTO_ID_FROM_MAC* the value will be the last 32 bit of the MAC address assigned to the *first* up and running interface of the host machine. **[range]** [0,0xffffffff]

6.242.3.4 System::UInt32 DDS::WireProtocolQosPolicy::rtps_instance_id

The RTPS Instance ID of the domain participant.

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::WireProtocolQosPolicy::RTPS_AUTO_ID (p. 330). The default value is interpreted as follows:

If **DDS::WireProtocolQosPolicy::rtps_auto_id_kind** (p. 1430) is equals to *RTPS_AUTO_ID_FROM_IP* (default value for this field) A counter is assigned that is incremented per new participant.

If **DDS::WireProtocolQosPolicy::rtps_auto_id_kind** (p. 1430) is equals to *RTPS_AUTO_ID_FROM_MAC* the first 24 bits are assigned to the application/task identifier and the last 8 bits are assigned to a counter incremented per new paricipant.

[range] [0,0xffffffff] **NOTE:** if we are using *DDS_RTPS_AUTO_ID_FROM_MAC* as **rtps_auto_id_kind** and you decide to manually handle the **rtps_instance_id** field, you have to ensure that both the two parts are different from zero otherwise the middleware will take responsability for them. We recommend to always specify the two parts separately in order to avoid errors. (`examples`)

6.242.3.5 RtpsWellKnownPorts_t DDS::WireProtocolQosPolicy::rtps_well_known_ports

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::RtpsWellKnownPorts_t::INTEROPERABLE_RTPS_WELL_KNOWN_PORTS** (p. 329)

6.242.3.6 System::Int32 DDS::WireProtocolQosPolicy::rtps_reserved_port_mask

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 domain participant is enabled. Failure to allocate a port that is computed based on the **DDS::RtpsWellKnownPorts_t** (p. 1142) will be detected at this time, and the enable operation will fail.

[default] DDS::RtpsReservedPortKind::RTPS_RESERVED_PORT_MASK_DEFAULT

6.242.3.7 WireProtocolQosPolicyAutoKind DDS::WireProtocolQosPolicy::rtps_auto_id_kind

Kind of auto mechanism used to calculate the GUID prefix.

[default] RTPS_AUTO_ID_FROM_IP

6.243 DDS::WriterDataLifecycleQosPolicy Struct Reference

Controls how a **DDS::DataWriter** (p. 499) handles the lifecycle of the instances (keys) that it is registered to manage.

`#include <managed_infrastructure.h>`

Static Public Member Functions

- static System::String^ **get_writerdatalifecycle_qos_policy_name** ()

  *Stringified human-readable name for **DDS::WriterDataLifecycleQosPolicy** (p. 1431).*

Properties

- System::Boolean **autodispose_unregistered_instances** `[get, set]`

  *Boolean flag that controls the behavior when the **DDS::DataWriter** (p. 499) unregisters an instance by means of the unregister operations.*

6.243.1 Detailed Description

Controls how a **DDS::DataWriter** (p. 499) handles the lifecycle of the instances (keys) that it is registered to manage.

Entity:

DDS::DataWriter (p. 499)

Properties:

RxO (p. 268) = N/A
Changeable (p. 269) = **YES** (p. 269)

6.243.2 Usage

This policy determines how the **DDS::DataWriter** (p. 499) 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. 499) or implicitly registered by directly writing the data).

Since the deletion of a **DataWriter** (p. 499) automatically unregisters all data instances it manages, the setting of the `autodispose_unregistered_instances` flag will only determine whether instances are ultimately disposed when the **DDS::DataWriter** (p. 499) is deleted either directly by means of the **DDS::Publisher::delete_datawriter** (p. 1056) operation or indirectly as a consequence of calling **DDS::Publisher::delete_contained_entities** (p. 1061) or **DDS::DomainParticipant::delete_contained_entities** (p. 638) that contains the **DataWriter** (p. 499).

You may use **DDS::TypedDataWriter::unregister_instance** (p. 1372) to indicate that the **DDS::DataWriter** (p. 499) no longer wants to send data for a **DDS::Topic** (p. 1258).

The behavior controlled by this QoS policy applies on a per instance (key) basis for keyed Topics, so that when a **DDS::DataWriter** (p. 499) unregisters an instance, RTI Data Distribution Service can automatically also dispose that instance. This is the default behavior.

In many cases where the ownership of a **Topic** (p. 1258) is shared (see **DDS::OwnershipQosPolicy** (p. 993)), DataWriters may want to relinquish their ownership of a particular instance of the **Topic** (p. 1258) 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** (p. 499) 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.

6.243.3 Property Documentation

6.243.3.1 System::Boolean DDS::WriterDataLifecycleQosPolicy::autodispose_unregistered_instances [get, set]

Boolean flag that controls the behavior when the **DDS::DataWriter** (p. 499) unregisters an instance by means of the unregister operations.

- true (default)

  The **DDS::DataWriter** (p. 499) will dispose 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.

- false

  The **DDS::DataWriter** (p. 499) will not dispose the instance. The application can still call one of the `dispose` operations prior to unregistering

the instance and accomplish the same effect.

[**default**] true

6.244 DDS::WstringSeq Class Reference

Instantiates `DDS::Sequence` (p. 1163) < System::Char* >.

`#include <managed_infrastructure.h>`

Inheritance diagram for DDS::WstringSeq::

Public Member Functions

- ^ **WstringSeq** ()

  Constructs an empty sequence of wide strings with an initial maximum of zero.

- ^ **WstringSeq** (System::Int32 max)

  Constructs an empty sequence of wide strings with the given initial maximum.

- ^ **WstringSeq** (**WstringSeq**^ strings)

  Constructs a new sequence containing the given wide strings.

6.244.1 Detailed Description

Instantiates `DDS::Sequence` (p. 1163) < System::Char* >.

Instantiates:

<<***generic***>> *(p. 175)* **DDS::Sequence** (p. 1163)

See also:

System::Char
DDS::StringSeq (p. 1192)
DDS::Sequence (p. 1163)

6.244.2 Constructor & Destructor Documentation

6.244.2.1 DDS::WstringSeq::WstringSeq () `[inline]`

Constructs an empty sequence of wide strings with an initial maximum of zero.

6.244.2.2 DDS::WstringSeq::WstringSeq (System::Int32 *max*) [inline]

Constructs an empty sequence of wide strings with the given initial maximum.

6.244.2.3 DDS::WstringSeq::WstringSeq (WstringSeq^ *strings*) [inline]

Constructs a new sequence containing the given wide strings.

Parameters:

strings the initial contents of this sequence

Chapter 7

Example Documentation

7.1 HelloWorld.cpp

7.1.1 Programming Language Type Description

The following programming language specific type representation is generated by **rtiddsgen** (p. 196) for use in application code, where:

- **Foo** (p. 877) = HelloWorld
- **FooSeq** (p. 880) = HelloWorldSeq

The following files are always generated in the C++/CLI language, even when code is generated with the `-language C#` option, because they depend on unmanaged code that ships with RTI Data Distribution Service. Once compiled, the code can be used from either C++/CLI or C# code; see the C# **publisher** (p. **??**) and **subscriber** (p. **??**) example code.

7.1.1.1 HelloWorld.h

[$(NDDSHOME)/example/CPPCLI/helloWorld/HelloWorld.h]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from HelloWorld.idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Data Distribution Service distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Data Distribution Service manual.
*/
```

```
#pragma once


struct DDS_TypeCode;


using namespace System;
using namespace DDS;



public ref struct HelloWorld
        : public DDS::ICopyable<HelloWorld^> {
    // --- Declared members: -----------------------------------------------
  public:

    System::String^ msg; // maximum length = (128)

    // --- Constructors and destructors: -----------------------------------
  public:
    HelloWorld();

    // --- Utility methods: ------------------------------------------------
  public:
    virtual System::Boolean copy_from(HelloWorld^ src);

    virtual System::Boolean Equals(System::Object^ other) override;


    static DDS::TypeCode^ get_typecode();

  private:
    static DDS::TypeCode^ _typecode;

}; // class HelloWorld




public ref class HelloWorldSeq sealed
        : public DDS::UserRefSequence<HelloWorld^> {
public:
    HelloWorldSeq() :
            DDS::UserRefSequence<HelloWorld^>() {
        // empty
    }
    HelloWorldSeq(System::Int32 max) :
            DDS::UserRefSequence<HelloWorld^>(max) {
        // empty
    }
    HelloWorldSeq(HelloWorldSeq^ src) :
            DDS::UserRefSequence<HelloWorld^>(src) {
        // empty
    }
};
```

```
DDS_TypeCode* HelloWorld_get_typecode();
```

7.1.1.2 HelloWorld.cpp

[$(NDDSHOME)/example/CPPCLI/helloWorld/HelloWorld.cpp]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from HelloWorld.idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Data Distribution Service distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Data Distribution Service manual.
*/


#pragma unmanaged
#include "ndds/ndds_cpp.h"
#pragma managed

using namespace System;
using namespace System::Collections;
using namespace DDS;


#include "HelloWorld.h"

/* ========================================================================= */
HelloWorld::HelloWorld() {

    msg = L"";

}

System::Boolean HelloWorld::copy_from(HelloWorld^ src) {
    HelloWorld^ dst = this;

    dst->msg = src->msg;

    return true;
}

Boolean HelloWorld::Equals(Object^ other) {
    if (other == nullptr) {
        return false;
    }
    if (this == other) {
        return true;
    }

    HelloWorld^ otherObj =
        dynamic_cast<HelloWorld^>(other);
    if (otherObj == nullptr) {
        return false;
    }
```

```
    if (!msg->Equals(otherObj->msg)) {
        return false;
    }

    return true;
}



DDS::TypeCode^ HelloWorld::get_typecode() {
    if (_typecode == nullptr) {
        _typecode = gcnew DDS::TypeCode(HelloWorld_get_typecode());
    }
    return _typecode;
}


DDS_TypeCode* HelloWorld_get_typecode()
{
    static RTIBool is_initialized = RTI_FALSE;

    static DDS_TypeCode HelloWorld_g_tc_msg_string = DDS_INITIALIZE_STRING_TYPECODE(128);

    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 */
            DDS_BOOLEAN_FALSE, /* Is a key? */
            DDS_PRIVATE_MEMBER,/* Ignored */
            0,/* Ignored */
            NULL/* Ignored */
        }
    };

    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 */
    }}; /* Type code for HelloWorld*/

    if (is_initialized) {
        return &HelloWorld_g_tc;
    }


    HelloWorld_g_tc_members[0]._representation._typeCode = (RTICdrTypeCode *)&HelloWorld_g_tc_msg_string;

    is_initialized = RTI_TRUE;

    return &HelloWorld_g_tc;
}
```

7.2 HelloWorld.idl

7.2.1 IDL Type Description

The data type to be disseminated by RTI Data Distribution Service is described in language independent IDL. The IDL file is input to **rtiddsgen** (p. 196), which produces the following files.

The programming language specific type representation of the type **Foo** (p. 877) = HelloWorld, for use in the application code.

- HelloWorld.h
- HelloWorld.cpp

User Data Type Support (p. 52) types as required by the DDS specification for use in the application code.

- HelloWorldSupport.h
- HelloWorldSupport.cpp

Methods required by the RTI Data Distribution Service implementation. These contains the auto-generated methods for serializing and deserializing the type.

- HelloWorldPlugin.h
- HelloWorldPlugin.cpp

The files above are always generated in the C++/CLI language, even when code is generated with the `-language C#` option, because they depend on unmanaged code that ships with RTI Data Distribution Service. Once compiled, the code can be used from either C++/CLI or C# code; see the C# **publisher** (p. **??**) and **subscriber** (p. **??**) example code.

7.2.1.1 HelloWorld.idl

[$(NDDSHOME)/example/CPPCLI/helloWorld/HelloWorld.idl]

```
struct HelloWorld {
    string<128> msg;
};
```

7.3 HelloWorld_publisher.cpp

7.3.1 RTI Data Distribution Service Publication Example

The publication example generated by **rtiddsgen** (p. 196). The example has been modified slightly to update the sample value.

7.3.1.1 HelloWorld_publisher.cpp

[$(NDDSHOME)/example/CPPCLI/helloWorld/HelloWorld_publisher.cpp]

```
/* HelloWorld_publisher.cpp

   A publication of data of type HelloWorld

   This file is derived from code automatically generated by the rtiddsgen
   command:

   rtiddsgen -language C++/CLI -example <arch> HelloWorld.idl

   Example publication of type HelloWorld automatically generated by
   'rtiddsgen'. To test them follow these steps:

   (1) Compile this file and the example subscription.

   (2) Start the subscription on the same domain used for RTI Data Distribution
       with the command
       HelloWorld_subscriber <domain_id> <sample_count>

   (3) Start the publication on the same domain used for RTI Data Distribution
       with the command
       HelloWorld_publisher <domain_id> <sample_count>

   (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 publishers and subscribers programs, and can
   add and remove them dynamically from the domain.


   Example:

       To run the example application on domain <domain_id>:

       HelloWorld_publisher <domain_id> <sample_count>
       HelloWorld_subscriber <domain_id> <sample_count>
*/

#ifndef IMPORT_HelloWorld
/* If this example code is packaged into an assembly other than that
 * containing the generated types themselves, no header inclusion is
 * necessary. In that case, simply define IMPORT_HelloWorld.
```

```
 */
#include "HelloWorldSupport.h"
#endif

using namespace System;


public ref class HelloWorldPublisher {
public:
    static void publish(int domain_id, int sample_count);

private:
    static void shutdown(
        DDS::DomainParticipant^ participant);
};

int main(array<System::String^>^ argv) {
    int domain_id = 0;
    if (argv->Length >= 1) {
        domain_id = Int32::Parse(argv[0]);
    }

    int sample_count = 0; /* infinite loop */
    if (argv->Length >= 2) {
        sample_count = Int32::Parse(argv[1]);
    }

    /* Uncomment this to turn on additional logging
    NDDS::ConfigLogger::get_instance()->set_verbosity_by_category(
        NDDS::LogCategory::NDDS_CONFIG_LOG_CATEGORY_API,
        NDDS::LogVerbosity::NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL);
    */
    try {
        HelloWorldPublisher::publish(
            domain_id, sample_count);
    }
    catch(DDS::Exception^) {
        return -1;
    }
    return 0;
}

void HelloWorldPublisher::publish(int domain_id, int sample_count) {
    /* To customize participant QoS, use
       DDS::DomainParticipantFactory::get_instance()->get_default_participant_qos() */
    DDS::DomainParticipant^ participant =
        DDS::DomainParticipantFactory::get_instance()->create_participant(
            domain_id,
            DDS::DomainParticipantFactory::PARTICIPANT_QOS_DEFAULT,
            nullptr /* listener */,
            DDS::StatusMask::STATUS_MASK_NONE);
    if (participant == nullptr) {
        shutdown(participant);
        throw gcnew ApplicationException("create_participant error");
    }

    /* To customize publisher QoS, use
```

```
   participant->get_default_publisher_qos() */
DDS::Publisher^ publisher = participant->create_publisher(
    DDS::DomainParticipant::PUBLISHER_QOS_DEFAULT,
    nullptr /* listener */,
    DDS::StatusMask::STATUS_MASK_NONE);
if (publisher == nullptr) {
    shutdown(participant);
    throw gcnew ApplicationException("create_publisher error");
}

/* Register type before creating topic */
System::String^ type_name = HelloWorldTypeSupport::get_type_name();
try {
    HelloWorldTypeSupport::register_type(
        participant, type_name);
} catch (DDS::Exception^ e) {
    shutdown(participant);
    throw e;
}

/* To customize topic QoS, use
   participant->get_default_topic_qos() */
DDS::Topic^ topic = participant->create_topic(
    "Example HelloWorld",
    type_name,
    DDS::DomainParticipant::TOPIC_QOS_DEFAULT,
    nullptr /* listener */,
    DDS::StatusMask::STATUS_MASK_NONE);
if (topic == nullptr) {
    shutdown(participant);
    throw gcnew ApplicationException("create_topic error");
}

/* To customize data writer QoS, use
   publisher->get_default_datawriter_qos() */
DDS::DataWriter^ writer = publisher->create_datawriter(
    topic,
    DDS::Publisher::DATAWRITER_QOS_DEFAULT,
    nullptr /* listener */,
    DDS::StatusMask::STATUS_MASK_NONE);
if (writer == nullptr) {
    shutdown(participant);
    throw gcnew ApplicationException("create_datawriter error");
}
HelloWorldDataWriter^ HelloWorld_writer =
    safe_cast<HelloWorldDataWriter^>(writer);

/* Create data sample for writing */
HelloWorld^ instance = HelloWorldTypeSupport::create_data();
if (instance == nullptr) {
    shutdown(participant);
    throw gcnew ApplicationException(
        "HelloWorldTypeSupport::create_data error");
}

/* For data type that has key, if the same instance is going to be
   written multiple times, initialize the key here
```

```
       and register the keyed instance prior to writing */
    DDS::InstanceHandle_t instance_handle = DDS::InstanceHandle_t::HANDLE_NIL;
/*
    instance_handle = HelloWorld_writer->register_instance(instance);
*/

    /* Main loop */
    const System::Int32 send_period = 4000; // milliseconds
    for (int count=0; (sample_count == 0) || (count < sample_count); ++count) {
        Console::WriteLine("Writing HelloWorld, count {0}", count);

        /* Modify the data to be sent here */
        instance->msg = "Hello World! (" + count + ")";

        try {
            HelloWorld_writer->write(instance, instance_handle);
        }
        catch(DDS::Exception ^e) {
            Console::WriteLine("write error: {0}", e);
        }

        System::Threading::Thread::Sleep(send_period);
    }

/*
    try {
        HelloWorld_writer->unregister_instance(
            instance, instance_handle);
    }
    catch(DDS::Exception ^e) {
        Console::WriteLine("unregister instance error: {0}", e);
    }
*/

    /* Delete data sample */
    try {
        HelloWorldTypeSupport::delete_data(instance);
    }
    catch(DDS::Exception ^e) {
        Console::WriteLine("HelloWorldTypeSupport::delete_data error: {0}", e);
    }

    /* Delete all entities */
    shutdown(participant);
}

/* Delete all entities */
void HelloWorldPublisher::shutdown(
        DDS::DomainParticipant^ participant) {

    if (participant != nullptr) {
        participant->delete_contained_entities();
        DDS::DomainParticipantFactory::get_instance()->delete_participant(participant);
    }

    /* 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. */
/*
    DDS::DomainParticipantFactory::finalize_instance();
*/
}
```

7.4 HelloWorld_publisher.cs

7.4.1 RTI Data Distribution Service Publication Example

The publication example generated by **rtiddsgen** (p. 196). The example has been modified slightly to update the sample value.

7.4.1.1 HelloWorld_publisher.cs

[$(NDDSHOME)/example/CSHARP/helloWorld/HelloWorld_publisher.cs]

```
/* HelloWorld_publisher.cs

   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 them follow these steps:

   (1) Compile this file and the example subscription.

   (2) Start the subscription on the same domain used for RTI Data Distribution
       with the command
       objs\<arch>\HelloWorld_subscriber <domain_id> <sample_count>

   (3) Start the publication on the same domain used for RTI Data Distribution
       with the command
       objs\<arch>\HelloWorld_publisher <domain_id> <sample_count>

   (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 publishers and subscribers programs, and can
   add and remove them dynamically from the domain.


   Example:

       To run the example application on domain <domain_id>:

       bin\<Debug|Release>\HelloWorld_publisher <domain_id> <sample_count>
       bin\<Debug|Release>\HelloWorld_subscriber <domain_id> <sample_count>


modification history
------------ -------
*/
```

```
using System;
using System.Collections.Generic;
using System.Text;

public class HelloWorldPublisher {

    public static void Main(string[] args) {

        // --- Get domain ID --- //
        int domain_id = 0;
        if (args.Length >= 1) {
            domain_id = Int32.Parse(args[0]);
        }

        // --- Get max loop count; 0 means infinite loop  --- //
        int sample_count = 0;
        if (args.Length >= 2) {
            sample_count = Int32.Parse(args[1]);
        }

        /* Uncomment this to turn on additional logging
        NDDS.ConfigLogger.get_instance().set_verbosity_by_category(
            NDDS.LogCategory.CONFIG_LOG_CATEGORY_API,
            NDDS.LogVerbosity.CONFIG_LOG_VERBOSITY_STATUS_ALL);
        */

        // --- Run --- //
        try {
            HelloWorldPublisher.publish(
                domain_id, sample_count);
        }
        catch(DDS.Exception)
        {
            Console.WriteLine("error in publisher");
        }
    }

    static void publish(int domain_id, int sample_count) {

        // --- Create participant --- //

        /* To customize participant QoS, use
           DDS.DomainParticipantFactory.get_instance().
               get_default_participant_qos() */
        DDS.DomainParticipant participant =
            DDS.DomainParticipantFactory.get_instance().create_participant(
                domain_id,
                DDS.DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT,
                null /* listener */,
                DDS.StatusMask.STATUS_MASK_NONE);
        if (participant == null) {
            shutdown(participant);
            throw new ApplicationException("create_participant error");
        }

        // --- Create publisher --- //
```

```
/* To customize publisher QoS, use
   participant.get_default_publisher_qos() */
DDS.Publisher publisher = participant.create_publisher(
DDS.DomainParticipant.PUBLISHER_QOS_DEFAULT,
null /* listener */,
DDS.StatusMask.STATUS_MASK_NONE);
if (publisher == null) {
    shutdown(participant);
    throw new ApplicationException("create_publisher error");
}

// --- Create topic --- //

/* Register type before creating topic */
System.String type_name = HelloWorldTypeSupport.get_type_name();
try {
    HelloWorldTypeSupport.register_type(
        participant, type_name);
}
catch(DDS.Exception e) {
    Console.WriteLine("register_type error {0}", e);
    shutdown(participant);
    throw e;
}

/* To customize topic QoS, use
   participant.get_default_topic_qos() */
DDS.Topic topic = participant.create_topic(
    "Example HelloWorld",
    type_name,
    DDS.DomainParticipant.TOPIC_QOS_DEFAULT,
    null /* listener */,
    DDS.StatusMask.STATUS_MASK_NONE);
if (topic == null) {
    shutdown(participant);
    throw new ApplicationException("create_topic error");
}

// --- Create writer --- //

/* To customize data writer QoS, use
    publisher.get_default_datawriter_qos() */
DDS.DataWriter writer = publisher.create_datawriter(
    topic,
    DDS.Publisher.DATAWRITER_QOS_DEFAULT,
    null /* listener */,
    DDS.StatusMask.STATUS_MASK_NONE);
if (writer == null) {
    shutdown(participant);
    throw new ApplicationException("create_datawriter error");
}
HelloWorldDataWriter HelloWorld_writer =
    (HelloWorldDataWriter)writer;

// --- Write --- //

/* Create data sample for writing */
```

```
        HelloWorld instance = HelloWorldTypeSupport.create_data();
        if (instance == null) {
            shutdown(participant);
            throw new ApplicationException(
                "HelloWorldTypeSupport.create_data error");
        }

        /* For data type that has key, if the same instance is going to be
           written multiple times, initialize the key here
           and register the keyed instance prior to writing */
        DDS.InstanceHandle_t instance_handle = DDS.InstanceHandle_t.HANDLE_NIL;
        /*
        instance_handle = HelloWorld_writer.register_instance(instance);
        */

        /* Main loop */
        const System.Int32 send_period = 4000; // milliseconds
        for (int count=0;
             (sample_count == 0) || (count < sample_count);
             ++count) {
            Console.WriteLine("Writing HelloWorld, count {0}", count);

            /* Modify the data to be sent here */
            instance.msg = "Hello World! (" + count + ")";

            try {
                HelloWorld_writer.write(instance, ref instance_handle);
            }
            catch(DDS.Exception e) {
                Console.WriteLine("write error {0}", e);
            }

            System.Threading.Thread.Sleep(send_period);
        }

        /*
        try {
            HelloWorld_writer.unregister_instance(
                instance, instance_handle);
        } catch(DDS.Exception e) {
            Console.WriteLine("unregister instance error: {0}", e);
        }
        */

        // --- Shutdown --- //

        /* Delete data sample */
        try {
            HelloWorldTypeSupport.delete_data(instance);
        } catch(DDS.Exception e) {
            Console.WriteLine(
                "HelloWorldTypeSupport.delete_data error: {0}", e);
        }

        /* Delete all entities */
        shutdown(participant);
    }
```

```
    static void shutdown(
        DDS.DomainParticipant participant) {

        /* Delete all entities */

        if (participant != null) {
            participant.delete_contained_entities();
            DDS.DomainParticipantFactory.get_instance().delete_participant(
                ref participant);
        }

        /* 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. */
        /*
        try {
            DDS.DomainParticipantFactory.finalize_instance();
        } catch (DDS.Exception e) {
            Console.WriteLine("finalize_instance error: {0}", e);
            throw e;
        }
        */
    }
}
```

7.5 HelloWorld_subscriber.cpp

7.5.1 RTI Data Distribution Service Subscription Example

The unmodified subscription example generated by **rtiddsgen** (p. 196).

7.5.1.1 HelloWorld_subscriber.cpp

[$(NDDSHOME)/example/CPPCLI/helloWorld/HelloWorld_subscriber.cpp]

```
/* HelloWorld_subscriber.cpp

   A subscription example

   This file is derived from code automatically generated by the rtiddsgen
   command:

   rtiddsgen -language C++/CLI -example <arch> HelloWorld.idl

   Example subscription of type HelloWorld automatically generated by
   'rtiddsgen'. To test them, follow these steps:

   (1) Compile this file and the example publication.

   (2) Start the subscription on the same domain used for RTI Data Distribution
       Service with the command
       HelloWorld_subscriber <domain_id> <sample_count>

   (3) Start the publication on the same domain used for RTI Data Distribution
       with the command
       HelloWorld_publisher <domain_id> <sample_count>

   (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 publishers and subscribers programs, and can
   add and remove them dynamically from the domain.

   Example:

       To run the example application on domain <domain_id>:

       HelloWorld_publisher <domain_id> <sample_count>
       HelloWorld_subscriber <domain_id> <sample_count>
*/

#ifndef IMPORT_HelloWorld
/* If this example code is packaged into an assembly other than that
 * containing the generated types themselves, no header inclusion is
 * necessary. In that case, simply define IMPORT_HelloWorld.
 */
```

```
#include "HelloWorldSupport.h"
#endif

using namespace System;


public ref class HelloWorldSubscriber {
public:
    static void subscribe(int domain_id, int sample_count);

private:
    static void shutdown(
        DDS::DomainParticipant^ participant);
};

public ref class HelloWorldListener : public DDS::DataReaderListener {
  public:
    virtual void on_requested_deadline_missed(
        DDS::DataReader^ /*reader*/,
        DDS::RequestedDeadlineMissedStatus% /*status*/) override {}

    virtual void on_requested_incompatible_qos(
        DDS::DataReader^ /*reader*/,
        DDS::RequestedIncompatibleQosStatus^ /*status*/) override {}

    virtual void on_sample_rejected(
        DDS::DataReader^ /*reader*/,
        DDS::SampleRejectedStatus% /*status*/) override {}

    virtual void on_liveliness_changed(
        DDS::DataReader^ /*reader*/,
        DDS::LivelinessChangedStatus% /*status*/) override {}

    virtual void on_sample_lost(
        DDS::DataReader^ /*reader*/,
        DDS::SampleLostStatus% /*status*/) override {}

    virtual void on_subscription_matched(
        DDS::DataReader^ /*reader*/,
        DDS::SubscriptionMatchedStatus% /*status*/) override {}

    virtual void on_data_available(DDS::DataReader^ reader) override;

    HelloWorldListener() {
        data_seq = gcnew HelloWorldSeq();
        info_seq = gcnew DDS::SampleInfoSeq();
    }

  private:
    HelloWorldSeq^ data_seq;
    DDS::SampleInfoSeq^ info_seq;
};

int main(array<System::String^>^ argv) {
    int domain_id = 0;
    if (argv->Length >= 1) {
        domain_id = Int32::Parse(argv[0]);
```

```
    }

    int sample_count = 0; /* infinite loop */
    if (argv->Length >= 2) {
        sample_count = Int32::Parse(argv[1]);
    }

    /* Uncomment this to turn on additional logging
    NDDS::ConfigLogger::get_instance()->set_verbosity_by_category(
        NDDS::LogCategory::NDDS_CONFIG_LOG_CATEGORY_API,
        NDDS::LogVerbosity::NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL);
    */

    try {
        HelloWorldSubscriber::subscribe(
            domain_id, sample_count);
    }
    catch(DDS::Exception^) {
        return -1;
    }
    return 0;
}

void HelloWorldSubscriber::subscribe(
        int domain_id, int sample_count) {

    /* To customize participant QoS, use
       the configuration file USER_QOS_PROFILES.xml */
    DDS::DomainParticipant^ participant =
        DDS::DomainParticipantFactory::get_instance()->create_participant(
            domain_id,
            DDS::DomainParticipantFactory::PARTICIPANT_QOS_DEFAULT,
            nullptr /* listener */,
            DDS::StatusMask::STATUS_MASK_NONE);
    if (participant == nullptr) {
        shutdown(participant);
        throw gcnew ApplicationException("create_participant error");
    }

    /* To customize subscriber QoS, use
       the configuration file USER_QOS_PROFILES.xml */
    DDS::Subscriber^ subscriber = participant->create_subscriber(
        DDS::DomainParticipant::SUBSCRIBER_QOS_DEFAULT,
        nullptr /* listener */,
        DDS::StatusMask::STATUS_MASK_NONE);
    if (subscriber == nullptr) {
        shutdown(participant);
        throw gcnew ApplicationException("create_subscriber error");
    }

    /* Register the type before creating the topic */
    System::String^ type_name = HelloWorldTypeSupport::get_type_name();
    try {
        HelloWorldTypeSupport::register_type(
            participant, type_name);
    } catch (DDS::Exception^ e) {
        shutdown(participant);
```

```
        throw e;
    }

    /* To customize topic QoS, use
       the configuration file USER_QOS_PROFILES.xml */
    DDS::Topic^ topic = participant->create_topic(
        "Example HelloWorld",
        type_name,
        DDS::DomainParticipant::TOPIC_QOS_DEFAULT,
        nullptr /* listener */,
        DDS::StatusMask::STATUS_MASK_NONE);
    if (topic == nullptr) {
        shutdown(participant);
        throw gcnew ApplicationException("create_topic error");
    }

    /* Create a data reader listener */
    HelloWorldListener^ reader_listener =
        gcnew HelloWorldListener();

    /* To customize the data reader QoS, use
       the configuration file USER_QOS_PROFILES.xml */
    DDS::DataReader^ reader = subscriber->create_datareader(
        topic,
        DDS::Subscriber::DATAREADER_QOS_DEFAULT,
        reader_listener,
        DDS::StatusMask::STATUS_MASK_ALL);
    if (reader == nullptr) {
        shutdown(participant);
        throw gcnew ApplicationException("create_datareader error");
    }

    /* Main loop */
    const System::Int32 receive_period = 4000; // milliseconds
    for (int count=0; (sample_count == 0) || (count < sample_count); ++count) {
        Console::WriteLine(
            "HelloWorld subscriber sleeping for {0} sec...",
            receive_period / 1000);

        System::Threading::Thread::Sleep(receive_period);
    }

    /* Delete all entities */
    shutdown(participant);
}

/* Delete all entities */
void HelloWorldSubscriber::shutdown(
        DDS::DomainParticipant^ participant) {

    if (participant != nullptr) {
        participant->delete_contained_entities();
        DDS::DomainParticipantFactory::get_instance()->delete_participant(
            participant);
    }

    /* RTI Data Distribution Service provides 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. */
/*
    DDS::DomainParticipantFactory::finalize_instance();
*/
}

void HelloWorldListener::on_data_available(DDS::DataReader^ reader) {
    HelloWorldDataReader^ HelloWorld_reader =
        safe_cast<HelloWorldDataReader^>(reader);

    try {
        HelloWorld_reader->take(
            data_seq,
            info_seq,
            DDS::ResourceLimitsQosPolicy::LENGTH_UNLIMITED,
            DDS::SampleStateKind::ANY_SAMPLE_STATE,
            DDS::ViewStateKind::ANY_VIEW_STATE,
            DDS::InstanceStateKind::ANY_INSTANCE_STATE);
    }
    catch(DDS::Retcode_NoData^) {
        return;
    }
    catch(DDS::Exception ^e) {
        Console::WriteLine("take error {0}", e);
        return;
    }

    System::Int32 data_length = data_seq->length;
    for (int i = 0; i < data_length; ++i) {
        if (info_seq->get_at(i)->valid_data) {
            HelloWorldTypeSupport::print_data(data_seq->get_at(i));
        }
    }

    try {
        HelloWorld_reader->return_loan(data_seq, info_seq);
    }
    catch(DDS::Exception ^e) {
        Console::WriteLine("return loan error {0}", e);
    }
}
```

7.6 HelloWorld_subscriber.cs

7.6.1 RTI Data Distribution Service Subscription Example

The unmodified subscription example generated by **rtiddsgen** (p. 196).

7.6.1.1 HelloWorld_subscriber.cs

[$(NDDSHOME)/example/CSHARP/helloWorld/HelloWorld_subscriber.cs]

```
using System;
using System.Collections.Generic;
using System.Text;
/* HelloWorld_subscriber.cs

   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 them, follow these steps:

   (1) Compile this file and the example publication.

   (2) Start the subscription on the same domain used for RTI Data Distribution
       Service with the command
       objs\<arch>\HelloWorld_subscriber <domain_id> <sample_count>

   (3) Start the publication on the same domain used for RTI Data Distribution
       with the command
       objs\<arch>\HelloWorld_publisher <domain_id> <sample_count>

   (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 publishers and subscribers programs, and can
   add and remove them dynamically from the domain.

   Example:

       To run the example application on domain <domain_id>:

       bin\<Debug|Release>\HelloWorld_publisher <domain_id> <sample_count>
       bin\<Debug|Release>\HelloWorld_subscriber <domain_id> <sample_count>


modification history
------------ -------
```

```
*/

public class HelloWorldSubscriber {

    public class HelloWorldListener : DDS.DataReaderListener {

        public override void on_requested_deadline_missed(
            DDS.DataReader reader,
            ref DDS.RequestedDeadlineMissedStatus status) {}

        public override void on_requested_incompatible_qos(
            DDS.DataReader reader,
            DDS.RequestedIncompatibleQosStatus status) {}

        public override void on_sample_rejected(
            DDS.DataReader reader,
            ref DDS.SampleRejectedStatus status) {}

        public override void on_liveliness_changed(
            DDS.DataReader reader,
            ref DDS.LivelinessChangedStatus status) {}

        public override void on_sample_lost(
            DDS.DataReader reader,
            ref DDS.SampleLostStatus status) {}

        public override void on_subscription_matched(
            DDS.DataReader reader,
            ref DDS.SubscriptionMatchedStatus status) {}

        public override void on_data_available(DDS.DataReader reader) {
            HelloWorldDataReader HelloWorld_reader =
                (HelloWorldDataReader)reader;

            try {
                HelloWorld_reader.take(
                    data_seq,
                    info_seq,
                    DDS.ResourceLimitsQosPolicy.LENGTH_UNLIMITED,
                    DDS.SampleStateKind.ANY_SAMPLE_STATE,
                    DDS.ViewStateKind.ANY_VIEW_STATE,
                    DDS.InstanceStateKind.ANY_INSTANCE_STATE);
            }
            catch(DDS.Retcode_NoData) {
                return;
            }
            catch(DDS.Exception e) {
                Console.WriteLine("take error {0}", e);
                return;
            }

            System.Int32 data_length = data_seq.length;
            for (int i = 0; i < data_length; ++i) {
                if (info_seq.get_at(i).valid_data) {
                    HelloWorldTypeSupport.print_data(data_seq.get_at(i));
                }
            }
```

```
            try {
                HelloWorld_reader.return_loan(data_seq, info_seq);
            }
            catch(DDS.Exception e) {
                Console.WriteLine("return loan error {0}", e);
            }
        }

        public HelloWorldListener() {
            data_seq = new HelloWorldSeq();
            info_seq = new DDS.SampleInfoSeq();
        }

        private HelloWorldSeq data_seq;
        private DDS.SampleInfoSeq info_seq;
    };

    public static void Main(string[] args) {

        // --- Get domain ID --- //
        int domain_id = 0;
        if (args.Length >= 1) {
            domain_id = Int32.Parse(args[0]);
        }

        // --- Get max loop count; 0 means infinite loop  --- //
        int sample_count = 0;
        if (args.Length >= 2) {
            sample_count = Int32.Parse(args[1]);
        }

        /* Uncomment this to turn on additional logging
        NDDS.ConfigLogger.get_instance().set_verbosity_by_category(
            NDDS.LogCategory.NDDS_CONFIG_LOG_CATEGORY_API,
            NDDS.LogVerbosity.NDDS_CONFIG_LOG_VERBOSITY_STATUS_ALL);
        */

        // --- Run --- //
        try {
            HelloWorldSubscriber.subscribe(
                domain_id, sample_count);
        }
        catch(DDS.Exception) {
            Console.WriteLine("error in subscriber");
        }
    }

    static void subscribe(int domain_id, int sample_count) {

        // --- Create participant --- //

        /* To customize the participant QoS, use
           the configuration file USER_QOS_PROFILES.xml */
        DDS.DomainParticipant participant =
            DDS.DomainParticipantFactory.get_instance().create_participant(
                domain_id,
```

```
        DDS.DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT,
        null /* listener */,
        DDS.StatusMask.STATUS_MASK_NONE);
if (participant == null) {
    shutdown(participant);
    throw new ApplicationException("create_participant error");
}

// --- Create subscriber --- //

/* To customize the subscriber QoS, use
   the configuration file USER_QOS_PROFILES.xml */
DDS.Subscriber subscriber = participant.create_subscriber(
    DDS.DomainParticipant.SUBSCRIBER_QOS_DEFAULT,
    null /* listener */,
    DDS.StatusMask.STATUS_MASK_NONE);
if (subscriber == null) {
    shutdown(participant);
    throw new ApplicationException("create_subscriber error");
}

// --- Create topic --- //

/* Register the type before creating the topic */
System.String type_name = HelloWorldTypeSupport.get_type_name();
try {
    HelloWorldTypeSupport.register_type(
        participant, type_name);
}
catch(DDS.Exception e) {
    Console.WriteLine("register_type error {0}", e);
    shutdown(participant);
    throw e;
}

/* To customize the topic QoS, use
   the configuration file USER_QOS_PROFILES.xml */
DDS.Topic topic = participant.create_topic(
    "Example HelloWorld",
    type_name,
    DDS.DomainParticipant.TOPIC_QOS_DEFAULT,
    null /* listener */,
    DDS.StatusMask.STATUS_MASK_NONE);
if (topic == null) {
    shutdown(participant);
    throw new ApplicationException("create_topic error");
}

// --- Create reader --- //

/* Create a data reader listener */
HelloWorldListener reader_listener =
    new HelloWorldListener();

/* To customize the data reader QoS, use
   the configuration file USER_QOS_PROFILES.xml */
DDS.DataReader reader = subscriber.create_datareader(
```

```
            topic,
            DDS.Subscriber.DATAREADER_QOS_DEFAULT,
            reader_listener,
            DDS.StatusMask.STATUS_MASK_ALL);
        if (reader == null) {
            shutdown(participant);
            reader_listener = null;
            throw new ApplicationException("create_datareader error");
        }

        // --- Wait for data --- //

        /* Main loop */
        const System.Int32 receive_period = 4000; // milliseconds
        for (int count=0;
             (sample_count == 0) || (count < sample_count);
             ++count) {
            Console.WriteLine(
                "HelloWorld subscriber sleeping for {0} sec...",
                receive_period / 1000);

            System.Threading.Thread.Sleep(receive_period);
        }

        // --- Shutdown --- //

        /* Delete all entities */
        shutdown(participant);
        reader_listener = null;
    }


    static void shutdown(
        DDS.DomainParticipant participant) {

        /* Delete all entities */

        if (participant != null) {
            participant.delete_contained_entities();
            DDS.DomainParticipantFactory.get_instance().delete_participant(
                ref participant);
        }

        /* RTI Data Distribution Service provides 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. */
        /*
        try {
            DDS.DomainParticipantFactory.finalize_instance();
        }
        catch(DDS.Exception e) {
            Console.WriteLine("finalize_instance error {0}", e);
            throw e;
        }
        */
    }
```

```
}
```

7.7 HelloWorldPlugin.cpp

7.7.1 RTI Data Distribution Service Implementation Support

Files generated by **rtiddsgen** (p. 196) that provided methods for type specific serialization and deserialization, to support the RTI Data Distribution Service implementation.

The following files are always generated in the C++/CLI language, even when code is generated with the `-language C#` option, because they depend on unmanaged code that ships with RTI Data Distribution Service. Once compiled, the code can be used from either C++/CLI or C# code; see the C# **publisher** (p. **??**) and **subscriber** (p. **??**) example code.

7.7.1.1 HelloWorldPlugin.h

[$(NDDSHOME)/example/CPPCLI/helloWorld/HelloWorldPlugin.h]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from HelloWorld.idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Data Distribution Service distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Data Distribution Service manual.
*/

#pragma once

#include "HelloWorld.h"



/* ------------------------------------------------------------------------
 * Type: HelloWorld
 * ------------------------------------------------------------------------ */

public ref class HelloWorldPlugin :
    DefaultTypePlugin<HelloWorld^> {
// --- Support methods: -------------------------------------------------------
public:
    void print_data(
        HelloWorld^ sample,
        System::String^ desc,
        System::UInt32 indent);


// --- (De)Serialize methods: -------------------------------------------------
public:
    virtual System::Boolean serialize(
        TypePluginDefaultEndpointData^ endpoint_data,
```

```
        HelloWorld^ sample,
        CdrStream% stream,
        System::Boolean serialize_encapsulation,
        System::UInt16  encapsulation_id,
        System::Boolean serialize_sample,
        System::Object^ endpoint_plugin_qos) override;

    virtual System::Boolean deserialize_sample(
        TypePluginDefaultEndpointData^ endpoint_data,
        HelloWorld^ sample,
        CdrStream% stream,
        System::Boolean deserialize_encapsulation,
        System::Boolean deserialize_sample,
        System::Object^ endpoint_plugin_qos) override;

    System::Boolean skip(
        TypePluginDefaultEndpointData^ endpoint_data,
        CdrStream% stream,
        System::Boolean skip_encapsulation,
        System::Boolean skip_sample,
        System::Object^ endpoint_plugin_qos);

    virtual System::UInt32 get_serialized_sample_max_size(
        TypePluginDefaultEndpointData^ endpoint_data,
        System::Boolean include_encapsulation,
        System::UInt16  encapsulation_id,
        System::UInt32 size) override;

    virtual System::UInt32 get_serialized_sample_min_size(
        TypePluginDefaultEndpointData^ endpoint_data,
        System::Boolean include_encapsulation,
        System::UInt16  encapsulation_id,
        System::UInt32 size) override;

    virtual System::UInt32 get_serialized_sample_size(
        TypePluginDefaultEndpointData^ endpoint_data,
        Boolean include_encapsulation,
        UInt16 encapsulation_id,
        UInt32 current_alignment,
        HelloWorld^ sample) override;

// ---  Key Management functions: ------------------------------------------
public:
    virtual System::UInt32 get_serialized_key_max_size(
        TypePluginDefaultEndpointData^ endpoint_data,
        System::Boolean include_encapsulation,
        System::UInt16  encapsulation_id,
        System::UInt32 current_alignment) override;

    virtual System::Boolean serialize_key(
        TypePluginDefaultEndpointData^ endpoint_data,
        HelloWorld^ key,
        CdrStream% stream,
        System::Boolean serialize_encapsulation,
        System::UInt16  encapsulation_id,
        System::Boolean serialize_sample,
        System::Object^ endpoint_plugin_qos) override;
```

```
    virtual System::Boolean deserialize_key_sample(
        TypePluginDefaultEndpointData^ endpoint_data,
        HelloWorld^ key,
        CdrStream% stream,
        System::Boolean deserialize_encapsulation,
        System::Boolean deserialize_sample,
        System::Object^ endpoint_plugin_qos) override;

    System::Boolean serialized_sample_to_key(
        TypePluginDefaultEndpointData^ endpoint_data,
        HelloWorld^ sample,
        CdrStream% stream,
        System::Boolean deserialize_encapsulation,
        System::Boolean deserialize_key,
        System::Object^ endpoint_plugin_qos);




// ---  Plug-in lifecycle management methods: --------------------------------
public:
    static HelloWorldPlugin^ get_instance();

    static void dispose();

private:
    HelloWorldPlugin()
            : DefaultTypePlugin(
                "HelloWorld",

                false, // not keyed

                false, // use RTPS-compliant alignment
                   HelloWorld::get_typecode()) {
        // empty
    }

    static HelloWorldPlugin^ _singleton;
};
```

7.7.1.2 HelloWorldPlugin.cpp

[$(NDDSHOME)/example/CPPCLI/helloWorld/HelloWorldPlugin.cpp]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from HelloWorld.idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Data Distribution Service distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Data Distribution Service manual.
*/

#include "HelloWorldPlugin.h"
```

```
#pragma unmanaged
#include "ndds/ndds_cpp.h"
#include "osapi/osapi_utility.h"
#pragma managed

using namespace System::Runtime::InteropServices;

/* ----------------------------------------------------------------------
 *  Type HelloWorld
 * ---------------------------------------------------------------------- */

/* ----------------------------------------------------------------------
    Support functions:
 * ---------------------------------------------------------------------- */

void
HelloWorldPlugin::print_data(
        HelloWorld^ sample,
        String^ desc,
        UInt32 indent_level) {

    for (UInt32 i = 0; i < indent_level; ++i) { Console::Write("   "); }

    if (desc != nullptr) {
        Console::WriteLine("{0}:", desc);
    } else {
        Console::WriteLine();
    }

    if (sample == nullptr) {
        Console::WriteLine("null");
        return;
    }


    DataPrintUtility::print_object(
        sample->msg, "msg", indent_level);

}


/* ----------------------------------------------------------------------
    (De)Serialize functions:
 * ---------------------------------------------------------------------- */

Boolean
HelloWorldPlugin::serialize(
    TypePluginDefaultEndpointData^ endpoint_data,
    HelloWorld^ sample,
    CdrStream% stream,
    Boolean serialize_encapsulation,
    UInt16 encapsulation_id,
    Boolean serialize_sample,
    Object^ endpoint_plugin_qos)
{
```

```
    CdrStreamPosition position;

    if (serialize_encapsulation) {
        /* Encapsulate sample */
        if (!stream.serialize_and_set_cdr_encapsulation(encapsulation_id)) {
            return false;
        }


        position = stream.reset_alignment();

    }

    if (serialize_sample) {

    if (!stream.serialize_string(sample->msg, (128))) {
        return false;
    }

    }


    if(serialize_encapsulation) {
        stream.restore_alignment(position);
    }


    return true;
}


Boolean
HelloWorldPlugin::deserialize_sample(
    TypePluginDefaultEndpointData^ endpoint_data,
    HelloWorld^ sample,
    CdrStream% stream,
    Boolean deserialize_encapsulation,
    Boolean deserialize_data,
    Object^ endpoint_plugin_qos)
{
    CdrStreamPosition position;

    if(deserialize_encapsulation) {
        /* Deserialize encapsulation */
        if (!stream.deserialize_and_set_cdr_encapsulation()) {
            return false;
        }


        position = stream.reset_alignment();

    }

    if (deserialize_data) {

    sample->msg = stream.deserialize_string();
    if (sample->msg == nullptr) {
```

```
        return false;
    }

    }


    if(deserialize_encapsulation) {
        stream.restore_alignment(position);
    }


    return true;
}


Boolean
HelloWorldPlugin::skip(
    TypePluginDefaultEndpointData^ endpoint_data,
    CdrStream% stream,
    Boolean skip_encapsulation,
    Boolean skip_sample,
    Object^ endpoint_plugin_qos)
{
    CdrStreamPosition position;


    if (skip_encapsulation) {
        if (!stream.skip_encapsulation()) {
            return false;
        }

        position = stream.reset_alignment();

    }

    if (skip_sample) {

    if (!stream.skip_string((128) + 1)) {
        return false;
    }

    }


    if(skip_encapsulation) {
        stream.restore_alignment(position);
    }


    return true;
}

/*
  size is the offset from the point where we have do a logical reset
  Return difference in size, not the final offset.
*/
UInt32
```

```
HelloWorldPlugin::get_serialized_sample_max_size(
    TypePluginDefaultEndpointData^ endpoint_data,
    Boolean include_encapsulation,
    UInt16 encapsulation_id,
    UInt32 current_alignment)
{

    UInt32 initial_alignment = current_alignment;

    UInt32 encapsulation_size = current_alignment;


    if (include_encapsulation) {
        if (!CdrStream::valid_encapsulation_id(encapsulation_id)) {
            return 1;
        }


        encapsulation_size = CdrSizes::ENCAPSULATION->serialized_size(
            current_alignment);
        encapsulation_size -= current_alignment;
        current_alignment = 0;
        initial_alignment = 0;

    }


    current_alignment +=  CdrSizes::STRING->serialized_size(
        current_alignment, (128) + 1);

    if (include_encapsulation) {
        current_alignment += encapsulation_size;
    }

    return current_alignment - initial_alignment;
}


UInt32
HelloWorldPlugin::get_serialized_sample_min_size(
    TypePluginDefaultEndpointData^ endpoint_data,
    Boolean include_encapsulation,
    UInt16 encapsulation_id,
    UInt32 current_alignment)
{

    UInt32 initial_alignment = current_alignment;

    UInt32 encapsulation_size = current_alignment;


    if (include_encapsulation) {
        if (!CdrStream::valid_encapsulation_id(encapsulation_id)) {
            return 1;
        }
```

```
        encapsulation_size = CdrSizes::ENCAPSULATION->serialized_size(
            encapsulation_size);
        current_alignment = 0;
        initial_alignment = 0;

    }


    current_alignment +=  CdrSizes::STRING->serialized_size(
        current_alignment, 1);

    if (include_encapsulation) {
        current_alignment += encapsulation_size;
    }

    return current_alignment - initial_alignment;
}


UInt32
HelloWorldPlugin::get_serialized_sample_size(
    TypePluginDefaultEndpointData^ endpoint_data,
    Boolean include_encapsulation,
    UInt16 encapsulation_id,
    UInt32 current_alignment,
    HelloWorld^ sample)
{

    UInt32 initial_alignment = current_alignment;

    UInt32 encapsulation_size = current_alignment;


    if (include_encapsulation) {
        if (!CdrStream::valid_encapsulation_id(encapsulation_id)) {
            return 1;
        }


        encapsulation_size = CdrSizes::ENCAPSULATION->serialized_size(
            current_alignment);
        encapsulation_size -= current_alignment;
        current_alignment = 0;
        initial_alignment = 0;

    }


    current_alignment += CdrSizes::STRING->serialized_size(current_alignment, sample->msg);

    if (include_encapsulation) {
        current_alignment += encapsulation_size;
    }

    return current_alignment - initial_alignment;
}
```

```
UInt32
HelloWorldPlugin::get_serialized_key_max_size(
    TypePluginDefaultEndpointData^ endpoint_data,
    Boolean include_encapsulation,
    UInt16 encapsulation_id,
    UInt32 current_alignment)
{

    UInt32 encapsulation_size = current_alignment;


    UInt32 initial_alignment = current_alignment;

    if (include_encapsulation) {
        if (!CdrStream::valid_encapsulation_id(encapsulation_id)) {
            return 1;
        }


        encapsulation_size = CdrSizes::ENCAPSULATION->serialized_size(
            current_alignment);
        current_alignment = 0;
        initial_alignment = 0;

    }


    current_alignment += get_serialized_sample_max_size(
        endpoint_data,false,encapsulation_id,current_alignment);

    if (include_encapsulation) {
        current_alignment += encapsulation_size;
    }

    return current_alignment - initial_alignment;
}

/* ----------------------------------------------------------------------
    Key Management functions:
 * ---------------------------------------------------------------------- */

Boolean
HelloWorldPlugin::serialize_key(
    TypePluginDefaultEndpointData^ endpoint_data,
    HelloWorld^ sample,
    CdrStream% stream,
    Boolean serialize_encapsulation,
    UInt16 encapsulation_id,
    Boolean serialize_key,
    Object^ endpoint_plugin_qos)
{
    CdrStreamPosition position;

    if (serialize_encapsulation) {
        /* Encapsulate sample */
        if (!stream.serialize_and_set_cdr_encapsulation(encapsulation_id)) {
```

```
            return false;
        }


        position = stream.reset_alignment();

    }

    if (serialize_key) {

        if (!serialize(
                endpoint_data,
                sample,
                stream,
                serialize_encapsulation,
                encapsulation_id,
                serialize_key,
                endpoint_plugin_qos)) {
            return false;
        }

    }


    if(serialize_encapsulation) {
        stream.restore_alignment(position);
    }


    return true;
}


Boolean HelloWorldPlugin::deserialize_key_sample(
    TypePluginDefaultEndpointData^ endpoint_data,
    HelloWorld^ sample,
    CdrStream% stream,
    Boolean deserialize_encapsulation,
    Boolean deserialize_key,
    Object^ endpoint_plugin_qos)
{
    CdrStreamPosition position;

    if (deserialize_encapsulation) {
        /* Deserialize encapsulation */
        if (!stream.deserialize_and_set_cdr_encapsulation()) {
            return false;
        }


        position = stream.reset_alignment();

    }

    if (deserialize_key) {

        if (!deserialize_sample(
```

```
                endpoint_data, sample, stream,
                deserialize_encapsulation,
                deserialize_key,
                endpoint_plugin_qos)) {
            return false;
        }

    }


    if(deserialize_encapsulation) {
        stream.restore_alignment(position);
    }


    return true;
}


Boolean
HelloWorldPlugin::serialized_sample_to_key(
    TypePluginDefaultEndpointData^ endpoint_data,
    HelloWorld^ sample,
    CdrStream% stream,
    Boolean deserialize_encapsulation,
    Boolean deserialize_key,
    Object^ endpoint_plugin_qos)
{
    CdrStreamPosition position;

    if(deserialize_encapsulation) {
        if (!stream.deserialize_and_set_cdr_encapsulation()) {
            return false;
        }

        position = stream.reset_alignment();

    }

    if (deserialize_key) {

        if (!deserialize_sample(
                endpoint_data,
                sample,
                stream,
                deserialize_encapsulation,
                deserialize_key,
                endpoint_plugin_qos)) {
            return false;
        }

    }


    if(deserialize_encapsulation) {
        stream.restore_alignment(position);
    }
```

```
    return true;
}



/* ---------------------------------------------------------------------
 * Plug-in Lifecycle Methods
 * --------------------------------------------------------------------- */

HelloWorldPlugin^
HelloWorldPlugin::get_instance() {
    if (_singleton == nullptr) {
        _singleton = gcnew HelloWorldPlugin();
    }
    return _singleton;
}


void
HelloWorldPlugin::dispose() {
    delete _singleton;
    _singleton = nullptr;
}
```

7.8 HelloWorldSupport.cpp

7.8.1 User Data Type Support

Files generated by **rtiddsgen** (p. 196) that implement the type specific APIs required by the DDS specification, as described in the **User Data Type Support** (p. 52), where:

- **FooTypeSupport** (p. 884) = HelloWorldTypeSupport
- **FooDataWriter** (p. 879) = HelloWorldDataWriter
- **FooDataReader** (p. 878) = HelloWorldDataReader

The following files are always generated in the C++/CLI language, even when code is generated with the `-language C#` option, because they depend on unmanaged code that ships with RTI Data Distribution Service. Once compiled, the code can be used from either C++/CLI or C# code; see the C# **publisher** (p. **??**) and **subscriber** (p. **??**) example code.

7.8.1.1 HelloWorldSupport.h

[$(NDDSHOME)/example/CPPCLI/helloWorld/HelloWorldSupport.h]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from HelloWorld.idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Data Distribution Service distribution.
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Data Distribution Service manual.
*/

#pragma once

#include "HelloWorld.h"


class DDSDataWriter;
class DDSDataReader;

// ------------------------------------------------------------------------
// HelloWorldTypeSupport
// ------------------------------------------------------------------------

ref class HelloWorldPlugin;

/* A collection of useful methods for dealing with objects of type
 * HelloWorld.
 */
```

```
public ref class HelloWorldTypeSupport
        : public DDS::TypedTypeSupport<HelloWorld^> {
    // --- Type name: ----------------------------------------------------
  public:
    static System::String^ TYPENAME = "HelloWorld";


    // --- Public Methods: -----------------------------------------------
  public:
    /* Get the default name of this type.
     *
     * An application can choose to register a type under any name, so
     * calling this method is strictly optional.
     */
    static System::String^ get_type_name();

    /* Register this type with the given participant under the given logical
     * name. This type must be registered before a Topic can be created that
     * uses it.
     */
    static void register_type(
            DDS::DomainParticipant^ participant,
            System::String^ type_name);

    /* Unregister this type from the given participant, where it was
     * previously registered under the given name. No further Topic creation
     * using this type will be possible.
     *
     * Unregistration allows some middleware resources to be reclaimed.
     */
    static void unregister_type(
            DDS::DomainParticipant^ participant,
            System::String^ type_name);

    /* Create an instance of the HelloWorld type.
     */
    static HelloWorld^ create_data();

    /* If instances of the HelloWorld type require any
     * explicit finalization, perform it now on the given sample.
     */
    static void delete_data(HelloWorld^ data);

    /* Write the contents of the data sample to standard out.
     */
    static void print_data(HelloWorld^ a_data);

    /* Perform a deep copy of the contents of one data sample over those of
     * another, overwriting it.
     */
    static void copy_data(
        HelloWorld^ dst_data,
        HelloWorld^ src_data);


    // --- Implementation: -----------------------------------------------
    /* The following code is for the use of the middleware infrastructure.
```

```
     * Applications are not expected to call it directly.
     */
public:
    virtual DDS::DataReader^ create_datareaderI(
        System::IntPtr impl) override;
    virtual DDS::DataWriter^ create_datawriterI(
        System::IntPtr impl) override;

private:
    static HelloWorldTypeSupport^ get_instance();

    HelloWorldTypeSupport();

private:
    static HelloWorldTypeSupport^ _singleton;
    HelloWorldPlugin^ _type_plugin;
};


// ---------------------------------------------------------------------------
// HelloWorldDataReader
// ---------------------------------------------------------------------------

public ref class HelloWorldDataReader :
        public DDS::TypedDataReader<HelloWorld^> {
    /* The following code is for the use of the middleware infrastructure.
     * Applications are not expected to call it directly.
     */
  internal:
    HelloWorldDataReader(System::IntPtr impl);
};


// ---------------------------------------------------------------------------
// HelloWorldDataWriter
// ---------------------------------------------------------------------------

public ref class HelloWorldDataWriter :
        public DDS::TypedDataWriter<HelloWorld^> {
    /* The following code is for the use of the middleware infrastructure.
     * Applications are not expected to call it directly.
     */
  internal:
    HelloWorldDataWriter(System::IntPtr impl);
};
```

7.8.1.2 HelloWorldSupport.cpp

[$(NDDSHOME)/example/CPPCLI/helloWorld/HelloWorldSupport.cpp]

```
/*
  WARNING: THIS FILE IS AUTO-GENERATED. DO NOT MODIFY.

  This file was generated from HelloWorld.idl using "rtiddsgen".
  The rtiddsgen tool is part of the RTI Data Distribution Service distribution.
```

```
  For more information, type 'rtiddsgen -help' at a command shell
  or consult the RTI Data Distribution Service manual.
*/

#include "HelloWorldSupport.h"
#include "HelloWorldPlugin.h"


using namespace System;
using namespace DDS;

/* ========================================================================= */

// ---------------------------------------------------------------------------
// HelloWorldDataWriter
// ---------------------------------------------------------------------------

HelloWorldDataWriter::HelloWorldDataWriter(
        System::IntPtr impl) : DDS::TypedDataWriter<HelloWorld^>(impl) {
    // empty
}


// ---------------------------------------------------------------------------
// HelloWorldDataReader
// ---------------------------------------------------------------------------

HelloWorldDataReader::HelloWorldDataReader(
        System::IntPtr impl) : DDS::TypedDataReader<HelloWorld^>(impl) {
    // empty
}


// ---------------------------------------------------------------------------
// HelloWorldTypeSupport
// ---------------------------------------------------------------------------

HelloWorldTypeSupport::HelloWorldTypeSupport()
        : DDS::TypedTypeSupport<HelloWorld^>(
            HelloWorldPlugin::get_instance()) {

    _type_plugin = HelloWorldPlugin::get_instance();
}

void HelloWorldTypeSupport::register_type(
        DDS::DomainParticipant^ participant,
        System::String^ type_name) {

    get_instance()->register_type_untyped(participant, type_name);
}

void HelloWorldTypeSupport::unregister_type(
        DDS::DomainParticipant^ participant,
        System::String^ type_name) {

    get_instance()->unregister_type_untyped(participant, type_name);
}
```

```
HelloWorld^ HelloWorldTypeSupport::create_data() {
    return gcnew HelloWorld();
}

void HelloWorldTypeSupport::delete_data(
        HelloWorld^ a_data) {
    /* If the generated type does not implement IDisposable (the default),
     * the following will no a no-op.
     */
    delete a_data;
}

void HelloWorldTypeSupport::print_data(HelloWorld^ a_data) {
     get_instance()->_type_plugin->print_data(a_data, nullptr, 0);
}

void HelloWorldTypeSupport::copy_data(
        HelloWorld^ dst, HelloWorld^ src) {

    get_instance()->copy_data_untyped(dst, src);
}

System::String^ HelloWorldTypeSupport::get_type_name() {
    return TYPENAME;
}

DDS::DataReader^ HelloWorldTypeSupport::create_datareaderI(
        System::IntPtr impl) {

    return gcnew HelloWorldDataReader(impl);
}

DDS::DataWriter^ HelloWorldTypeSupport::create_datawriterI(
        System::IntPtr impl) {

    return gcnew HelloWorldDataWriter(impl);
}

HelloWorldTypeSupport^
HelloWorldTypeSupport::get_instance() {
    if (_singleton == nullptr) {
        _singleton = gcnew HelloWorldTypeSupport();
    }
    return _singleton;
}
```