
CoreDX ™ Data Distribution Service

The leading Small Footprint DDS Middleware

# C# Reference Manual

Twin Oaks Computing, Inc

Castle Rock, CO 80108

Jan 2017




## Trademarks



## Copy and Use Restrictions



## DISCLAIMER OF WARRANTY



## Contact

Twin Oaks Computing, Inc
755 Maleta Ln, Ste 203
Castle Rock, CO 80108
(720) 733-7906
contact@twinoakscomputing.com
http://www.twinoakscomputing.com
http://twitter.com/CoreDX_DDS

# Contents

# Chapter 1

# Introduction

Welcome to the CoreDX DDS for C# API documentation from Twin Oaks Computing, Inc.

## Introduction

CoreDX DDS is a small-footprint, high-performance communications middleware compliant with the OMG Data Distribution Service (DDS) standard. CoreDX DDS supports multiple hardware architectures and operating systems, and is intended to facilitate the development of robust, near real-time, highly distributed systems.

This is the **CoreDX DDS for C# Reference Manual**. It provides a detailed reference for the CoreDX Data Distribution Service implementation from Twin Oaks Computing, Inc. The manual includes documentation on all of the CoreDX DDS data types and Application Programming Interface (API) routines.

The **CoreDX DDS Programmers Guide** provides more information on using the CoreDX DDS API and related tools to produce a complete DDS enabled application.

The CoreDX DDS software provides a high-throughput, standards compliant, data communications infrastructure. CoreDX DDS offers the tools you need to realize Open Architecture goals. Built with a focus on performance, the CoreDX DDS software delivers a quality implementation of the OMG Data Distribution Service (DDS) standard.

The CoreDX DDS software implements the essential Data-Centric Publish-Subscribe (DCPS) communications layer as documented in the OMG DDS Standard. This standalone package, provides everything needed to integrate QoS enabled, Publish-Subscribe messaging into an application. The core software is written in the C language, and is optimized to be small and fast. The core package includes C and C++ language bindings for application integration. This reference manual describes the CoreDX DDS "C#" language binding.

## Contents

The reference documentation includes information on the following API constructs:

1. Entities.This includes the primary objects with which an application must interact to enable DDS publish-subscribe communications.
    - DomainParticipantFactory
    - DomainParticipant
    - Topic
    - ContentFilteredTopic

- MultiTopic
- Publisher
- Subscriber
- DataReader
- DataWriter

2. Quality of Service. This section documents the Quality of Service (QoS) structures that configure the behavior of the CoreDX DDS middleware.
   - DomainParticipantFactoryQos
   - DomainParticipantQos
   - TopicQos
   - PublisherQos
   - SubscriberQos
   - DataReaderQos
   - DataWriterQos
3. Listeners and Events. This section covers the various structures and concepts involved in delivering events to the application.
   - DomainParticipantListener
   - TopicListener
   - PublisherListener
   - SubscriberListener
   - DataReaderListener
   - DataWriterListener
4. Status. This section describes the types of status maintained by the infrastructure.
   - InconsistentTopicStatus
   - LivelinessChangedStatus
   - LivelinessLostStatus
   - OfferedDeadlineMissedStatus
   - OfferedIncompatibleQosStatus
   - PublicationMatchedStatus
   - RequestedDeadlineMissedStatus
   - RequestedIncompatibleQosStatus
   - SampleLostStatus
   - SampleRejectedStatus
   - SubscriptionMatchedStatus
5. RPC over DDS
   - ClientEndpoint
   - ClientParams
   - ListenerBase
   - Replier
   - ReplierBase

- ReplierListener
- ReplierParams
- Requester
- RequesterListener
- RequesterParams
- RPCEntity
- ServiceEndpoint
- ServiceParams
- ServiceProxy
- SimpleReplierListener
- SimpleRequesterListener

6. Transports. This section includes Transport objects.
   - UDP Transport
   - TCP Transport
   - LMT Transport (local machine)
7. Miscellaneous. This section includes miscellaneous support objects.
   - Condition
   - GuardCondition
   - StatusCondition
   - QueryCondition
   - WaitSet

## Intended Audience

This document is intended for software developers who are integrating the CoreDX DDS software into their application(s). The reference manual assumes that the reader is competent in programming languages and software development concepts. CoreDX DDS supports multiple languages, and this reference manual focuses on the C# programming language.

## 1.1 DDS Builtin Entities and Data Types

### 1.1.1 Description

#### Classes

- class ParticipantBuiltinTopicDataDataReader
- class PublicationBuiltinTopicDataDataReader
- class SubscriptionBuiltinTopicDataDataReader

## 1.2 DDS Conditions

### 1.2.1 Description

#### Classes

- class Condition
- class GuardCondition
- class ReadCondition
- class QueryCondition

## 1.3 CoreDX DDS DynamicType

## 1.4 DDS Entities

### 1.4.1 Description

#### Classes

- class FooDataReader
- class FooDataWriter
- class DomainParticipant
- class DomainParticipantFactory
- class DataReader
- class DataWriter
- class Entity
- class DomainEntity
- class Publisher
- class Subscriber
- class Topic
- class ContentFilteredTopic

## 1.5 DDS Conditions, Listeners, and WaitSets

### 1.5.1 Description

#### Modules

- DDS Listeners
- DDS Conditions
- DDS WaitSets

## 1.6 DDS Listeners

### 1.6.1 Description

#### Classes

- class DomainParticipantListener
- class TopicListener
- class PublisherListener
- class DataWriterListener
- class SubscriberListener
- class DataReaderListener

## 1.7 DDS Quality of Service

### 1.7.1 Description

#### Classes

- enum DurabilityQosPolicyKind
- enum PresentationQosPolicyAccessScopeKind
- enum OwnershipQosPolicyKind
- enum LivelinessQosPolicyKind
- enum ReliabilityQosPolicyKind
- enum DestinationOrderQosPolicyKind
- enum HistoryQosPolicyKind
- enum TypeConsistencyKind
- enum DiscoveryQosPolicyDiscoveryKind
- struct ThreadModelQosPolicy
- class DomainParticipantFactoryQos
- class DomainParticipantQos
- class TopicQos
- class PublisherQos
- class SubscriberQos
- class DataWriterQos
- class DataReaderQos
- class QosProvider

  *QosProvider loads QoS settings from a library and provides interfaces to access entity specific QoS Policies.*

#### Functions

- QosProvider (string uri, string profile)

  *Constructor. Creates a new QosProvider instance that loads QoS settings from a library.*
- void cleanup ()

  *"Destructor". Returns a QosProvider back to the factory.*
- DomainParticipantFactoryQos get_participantfactory_qos (string id)

  *Access a DomainParticipantFactory QoS.*
- DomainParticipantQos get_participant_qos (string id)

  *Access a DomainParticipant QoS.*
- TopicQos get_topic_qos (string id)

  *Access a Topic QoS.*
- SubscriberQos get_subscriber_qos (string id)

  *Access a Subscriber QoS.*
- PublisherQos get_publisher_qos (string id)

  *Access a Publisher QoS.*
- DataReaderQos get_datareader_qos (string id)

  *Access a DataReader QoS.*
- DataWriterQos get_datawriter_qos (string id)

  *Access a DataWriter QoS.*

### 1.7.2 Function Documentation

#### 1.7.2.1 void cleanup ( ) [inline]

"Destructor". Returns a QosProvider back to the factory.

This will reclaim any resources allocated by the QosProvider.

#### 1.7.2.2 DataReaderQos get_datareader_qos ( string *id* ) [inline]

Access a DataReader QoS.

If 'id' is NULL, then the first instance of DataReaderQos in the library is returned. If 'id' is not NULL, then the 'name' field of the QoS policy is checked, and the first match is returned.

#### 1.7.2.3 DataWriterQos get_datawriter_qos ( string *id* ) [inline]

Access a DataWriter QoS.

If 'id' is NULL, then the first instance of DataWriterQos in the library is returned. If 'id' is not NULL, then the 'name' field of the QoS policy is checked, and the first match is returned.

#### 1.7.2.4 DomainParticipantQos get_participant_qos ( string *id* ) [inline]

Access a DomainParticipant QoS.

If 'id' is NULL, then the first instance of DomainParticipantQos in the library is returned. If 'id' is not NULL, then the 'name' field of the QoS policy is checked, and the first match is returned.

#### 1.7.2.5 DomainParticipantFactoryQos get_participantfactory_qos ( string *id* ) [inline]

Access a DomainParticipantFactory QoS.

If 'id' is NULL, then the first instance of DomainParticipantFactoryQos in the library is returned. If 'id' is not NULL, then the 'name' field of the DomainParticpantFactoryQos is checked, and the first match is returned.

#### 1.7.2.6 PublisherQos get_publisher_qos ( string *id* ) [inline]

Access a Publisher QoS.

If 'id' is NULL, then the first instance of PublisherQos in the library is returned. If 'id' is not NULL, then the 'name' field of the QoS policy is checked, and the first match is returned.

#### 1.7.2.7 SubscriberQos get_subscriber_qos ( string *id* ) [inline]

Access a Subscriber QoS.

If 'id' is NULL, then the first instance of SubscriberQos in the library is returned. If 'id' is not NULL, then the 'name' field of the QoS policy is checked, and the first match is returned.

#### 1.7.2.8 TopicQos get_topic_qos ( string *id* ) [inline]

Access a Topic QoS.

If 'id' is NULL, then the first instance of TopicQos in the library is returned. If 'id' is not NULL, then the 'name' field of the QoS policy is checked, and the first match is returned.

#### 1.7.2.9 QosProvider ( string *uri,* string *profile* ) [inline]

Constructor. Creates a new QosProvider instance that loads QoS settings from a library.

The QoS library source is indicated by 'uri'. Currently CoreDX DDS supports only the 'file:///' type URI. The QosProvider selects QoS policies from the library based on the 'profile' parameter.

## 1.8 CoreDX DDS RPC

### 1.8.1 Description

#### Classes

- class ClientEndpoint< TReq, TRep >
- class ClientParams
- interface ReplierBase
- class Replier< TReq, TRep >
- class ReplierParams
- class Requester< TReq, TRep >
- class RequesterParams
- interface RPCEntity
- interface ListenerBase
- interface SimpleReplierListener< TReq, TRep >
- interface ReplierListener< TReq, TRep >
- interface SimpleRequesterListener< TRep >
- interface RequesterListener< TReq, TRep >
- enum ServiceStatus
- class ServiceEndpoint
- class ServiceParams
- class ServiceProxy< TReq, TRep >

## 1.9 DDS Status Structures

### 1.9.1 Description

- InconsistentTopicStatus
- LivelinessChangedStatus
- LivelinessLostStatus
- OfferedDeadlineMissedStatus
- OfferedIncompatibleQosStatus
- PublicationMatchedStatus
- RequestedDeadlineMissedStatus
- RequestedIncompatibleQosStatus
- SampleLostStatus
- SampleRejectedStatus
- _SubscriptionMatchedStatus

#### Classes

- class InconsistentTopicStatus
- class OfferedDeadlineMissedStatus
- class OfferedIncompatibleQosStatus
- class LivelinessLostStatus
- class PublicationMatchedStatus
- class RequestedDeadlineMissedStatus
- class RequestedIncompatibleQosStatus
- enum SampleRejectedStatusKind
- class SampleRejectedStatus
- class LivelinessChangedStatus
- class SubscriptionMatchedStatus
- class SampleLostStatus
- class CacheStatistics

## 1.10 CoreDX DDS Tranpsorts

### 1.10.1 Description

#### Classes

- class Transport
- class IpTransportInterface
- class UdpTransport
- class UdpTransportConfig
- class TcpTransport
- class TcpTransportConfig
- class SslTransport
- class SslTransportConfig
- class LmtTransportConfig
- class LmtTransport

## 1.11 DDS WaitSets

### 1.11.1 Description

#### Classes

- class WaitSet

# Chapter 2

# Primary DDS Entities and Types

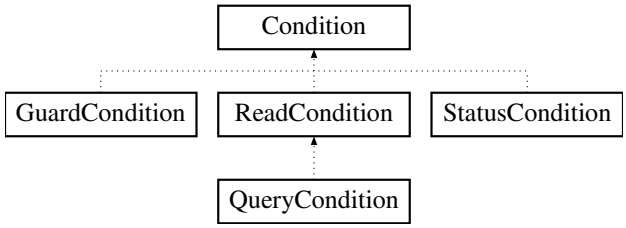
This section documents the DDS Entities and Types that the application developer will almost always interact with.

## 2.1 Condition Class Reference

### 2.1.1 Description

A Condition can be added to a WaitSet to provide synchronous event notification.

A Condition has a **trigger_value** which can be true or false. Inheritance diagram for Condition:




#### Public Member Functions

- bool get_trigger_value ()

### 2.1.2 Member Function Documentation

#### 2.1.2.1 bool get_trigger_value ( ) [inline]

This routine returns the current value of the **trigger_value** in Condition **c.**

A non-zero return value indicates that the **trigger_value** is TRUE.

A zero return value indicates that the **trigger_value** is FALSE.

## 2.2 ContentFilteredTopic Class Reference

### 2.2.1 Description

ContentFilteredTopic provides a topic that may include data filtered from a related Topic. The ContentFilteredTopic is associated with another un-filtered topic **related_topic**. It applies a filter to the data of the related topic. If a data sample passes the filter, it will be made available to a DataReader associated with the ContentFilteredTopic.

The **filter_expression** is an SQL like condition expression, and **filter_parameters** provide optional parameters that are referenced by the **filter_expression**. The syntax of the filter expression is similar to the WHERE clause in SQL. For example "x<4" is a valid filter expression. It would test that data member 'x' is less than value '4'. If the filter expression evaluates to TRUE, then the data sample will be available, otherwise the data sample would be 'filtered' (excluded).

CoreDX DDS supports the '**LIKE**' operator (and NOT LIKE) for regular expression string matching. The pattern string in a LIKE clause can contain '' to match zero or more characters, '_' to match a single character, or '[<characters>]' to match a range of characters.

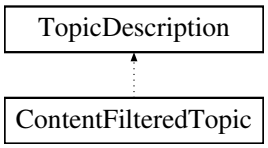
CoreDX DDS also includes support for the '**IN**' operator. This provides a very powerful mechanism for testing that a value appears in a set of values. For example "symbol IN ('ge', 'msft', 'ibm')" will select all samples that have a symbol value of 'ge', 'msft', or 'ibm'. This could also be written as a series of equality tests combined with the OR operator; however, the IN operator is much more efficient. A filter that matches on several hundred or even thousands of values can be implemented very efficiently using the 'IN' operator.

The filter_expression can refer to parameters. The syntax for paramters is the percent sign '' followed by a number. The number is the index of the paramter in the **filter_paramters** sequence. Parameters are counted starting at zero. So, "%0" refers to the first parameter, and "%4" refers to the fifth paramter. Using this syntax, the expression "x<%0" would test the value of 'x' against the first parameter in the sequence.

**See also**

DDS.DomainParticipant.create_contentfilteredtopic()

Inheritance diagram for ContentFilteredTopic:




### Public Member Functions

- DomainParticipant get_participant ()
- String get_type_name ()
- String get_name ()
- Topic get_related_topic ()
- ReturnCode_t get_expression_parameters (List< String > eparams)
- ReturnCode_t set_expression_parameters (List< String > filter_parameters)

### 2.2.2 Member Function Documentation

#### 2.2.2.1 ReturnCode_t get_expression_parameters ( List< String > *eparams* ) [inline]

This accesses the current set of parameters used by the ContentFilteredTopic.

The **parameters** String Sequence is populated with the current set of parameters.

#### 2.2.2.2 String get_name ( ) [inline]

This operation returns **topic_name** of the Topic.

Implements TopicDescription.

#### 2.2.2.3 DomainParticipant get_participant ( ) [inline]

This operation returns the parent **DomainParticipant** of the Topic.

Implements TopicDescription.

#### 2.2.2.4 Topic get_related_topic ( ) [inline]

This returns the real Topic associated with the ContentFilteredTopic.

That is, the Topic provided when the ContentFilteredTopic was created.

#### 2.2.2.5 String get_type_name ( ) [inline]

This operation returns **type_name** of the Topic.

Implements TopicDescription.

#### 2.2.2.6 ReturnCode_t set_expression_parameters ( List< String > *filter_parameters* ) [inline]

This specifies a new set of parameters for use with the filter_expression.

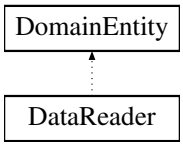
The **filter_expression** is an SQL like condition expression, and the **parameters** argument provides optional parameters that are referenced by the **filter_expression**. The syntax for refering to paramters in a filter_expression is the percent sign '' followed by a number. The number is the index of the paramter in the **filter_paramters** sequence. Parameters are counted starting at zero. So, "%0" refers to the first parameter, and "%4" refers to the fifth paramter. Using this syntax, the expression "x<%0" would test the value of 'x' against the first parameter in the sequence.

## 2.3 DataReader Class Reference

### 2.3.1 Description

The DataReader entity allows the application to subscribe to and read data.

The DataReader is an abstract class that is extended to support a particular data type required by the application. A DataReader is associated with a single TopicDescription (Topic, MultiTopic, or ContentFilteredTopic). Inheritance diagram for DataReader:




#### Public Member Functions

- new ReturnCode_t enable ()
- override InstanceHandle_t get_instance_handle ()
- ReadCondition create_readcondition (uint sample_states, uint view_states, uint instance_states)
- QueryCondition create_querycondition (uint sample_states, uint view_states, uint instance_states, String query_expression, List< String > query_parameters)
- ReturnCode_t delete_readcondition (ReadCondition rc)
- ReturnCode_t delete_contained_entities ()
- ReturnCode_t set_qos (DataReaderQos qos)
- ReturnCode_t get_qos (DataReaderQos qos)
- ReturnCode_t set_listener (DataReaderListener new_listener, uint mask)
- DataReaderListener get_listener ()
- TopicDescription get_topicdescription ()
- Subscriber get_subscriber ()
- ReturnCode_t get_sample_rejected_status (out SampleRejectedStatus status)
- ReturnCode_t get_liveliness_changed_status (out LivelinessChangedStatus status)
- ReturnCode_t get_requested_deadline_missed_status (out RequestedDeadlineMissedStatus status)
- ReturnCode_t get_requested_incompatible_qos_status (out RequestedIncompatibleQosStatus status)
- ReturnCode_t get_subscription_matched_status (out SubscriptionMatchedStatus status)
- ReturnCode_t get_sample_lost_status (out SampleLostStatus status)
- ReturnCode_t wait_for_historical_data (Duration_t max_wait)
- ReturnCode_t get_matched_publications (List< InstanceHandle_t > publication_handles)
- ReturnCode_t get_matched_publication_data (PublicationBuiltinTopicData publication_data, InstanceHandle_t publication_handle)
- ReturnCode_t get_guid (GUID_t g)

### 2.3.2 Member Function Documentation

#### 2.3.2.1 QueryCondition create_querycondition ( uint *sample_states,* uint *view_states,* uint *instance_states,* String *query_expression,* List< String > *query_parameters* ) [inline]

Creates a DDS.QueryCondition.

The returned QueryCondition can be used as an argument to read_w_condition() or take_w_condition().

The **query_expression** is an SQL like condition expression, and **query_parameters** provide optional parameters that are referenced by the **query_expression**. The syntax of the query expression is similar to the WHERE clause in SQL. For example "x<4" is a valid query expression. It would test that data member 'x' is less than value '4'. If the query expression evaluates to TRUE, then the data sample will be available, otherwise the data sample will be 'filtered' (excluded).

CoreDX DDS supports the '**LIKE**' operator (and NOT LIKE) for regular expression string matching. The pattern string in a LIKE clause can contain '' to match zero or more characters, '_' to match a single character, or '[<characters>]' to match a range of characters.

CoreDX DDS also includes support for the '**IN**' operator. This provides a very powerful mechanism for testing that a value appears in a set of values. For example "symbol IN ('ge', 'msft', 'ibm')" will select all samples that have a symbol value of 'ge', 'msft', or 'ibm'. This could also be written as a series of equality tests combined with the OR operator; however, the IN operator is much more efficient. A query that matches on several hundred or even thousands of values can be implemented very efficiently using the 'IN' operator.

The query_expression can refer to parameters. The syntax for paramters is the percent sign '' followed by a number. The number is the index of the paramter in the **query_paramters** sequence. Parameters are counted starting at zero. So, "%0" refers to the first parameter, and "%4" refers to the fifth paramter. Using this syntax, the expression "x<%0" would test the value of 'x' against the first parameter in the sequence.

**See also**

FooDataReader::read_w_condition().
FooDataReader::take_w_condition().

**Not Yet Supported** QueryConditions are not yet supported as triggers for a WaitSet.

#### 2.3.2.2 ReadCondition create_readcondition ( uint *sample_states,* uint *view_states,* uint *instance_states* ) [inline]

Creates a ReadCondition that is associated with this DataReader. The returned condition can be added to a WaitSet or used in a call to the specialized **read()** or **take()** operations. For example see FooDataReader.read_w_condition()

#### 2.3.2.3 ReturnCode_t delete_contained_entities ( ) [inline]

This operation deletes all the ReadCondition and QueryCondition objects previously created by means of the DataReader.create_readcondition() and DataReader.create_querycondition() operations.

After successful execution, the application may delete the Publisher by calling Subscriber.delete_datareader().

If any of the objects cannot be deleted, this routine will return DDS.RETCODE_PRECONDITION_NOT_MET.

#### 2.3.2.4 ReturnCode_t delete_readcondition ( ReadCondition *rc* ) [inline]

Destroys a ReadCondition (or QueryCondition). The provided **a_condition** must have been previously created via a call to DataReader.create_readcondition() or DataReader.create_querycondition().

The **a_condition** argument must belong to DataReader **dr.** Otherwise, the error DDS.RETCODE_PRECONDITION_NOT_MET will be returned.

If the DataReader is actively processing the ReadCondition, this routine will return DDS.RETCODE_ERROR; in this case, the delete_readcondition() call should be re-tried.

#### 2.3.2.5 new ReturnCode_t enable ( ) [inline], [virtual]

Enables the DataReader. A DataReader is created either enabled or not based on the SubscriberQos setting **entity_factory.** When a DataReader is not enabled, only the following sub-set of all DataReader operations are legal:

- operations to get and set QoS policies,
- get_statuscondition(),
- get_status_changes(),

Any other operation may return the DDS.RETCODE_NOT_ENABLED error. DataReader_enable() may be called on an already enabled DataReader [it will have no effect].

Reimplemented from Entity.

#### 2.3.2.6 ReturnCode_t get_guid ( GUID_t *g* ) [inline]

Access the GUID which uniquely identifies this reader.

#### 2.3.2.7 override InstanceHandle_t get_instance_handle ( ) [inline], [virtual]

Gets the handle that locally identifies this Entity.

Reimplemented from Entity.

#### 2.3.2.8 DataReaderListener get_listener ( ) [inline]

This operation returns the currently installed DataReaderListener.

#### 2.3.2.9 ReturnCode_t get_liveliness_changed_status ( out LivelinessChangedStatus *status* ) [inline]

Provides access to the current LivelinessChangedStatus of the DataReader. As a side-effect, this routine will reset the **total_count_change** status field to zero.

#### 2.3.2.10 ReturnCode_t get_matched_publication_data ( PublicationBuiltinTopicData *publication_data,* InstanceHandle_t *publication_handle* ) [inline]

This operation returns data that describes a particular matched DataWriter identified by **publication_handle.** An appropriate handle can be obtained through a call to DataReader.get_matched_publications().

If **publication_handle** does not identify a currently matched DataWriter, this routine will return DDS.RETCODE_PRECONDITION_NOT_M

#### 2.3.2.11 ReturnCode_t get_matched_publications ( List< InstanceHandle_t > *publication_handles* ) [inline]

This operation retrieves the list of DataWriters currently matched with this DataReader **dr.** This list will include the handles that identify DataWriters which have matching Topic and compatible QoS with DataReader.

If a DataWriter has been ignored by a call to DomainParticipant.ignore_publication(), then it will not appear in the list.

**Parameters**

| | |
|---|---|
| *publication_handles* | A vector that will be populated with InstanceHandle_t(s). |

#### 2.3.2.12 ReturnCode_t get_qos ( DataReaderQos *qos* ) [inline]

Returns the current DataReaderQos settings held in the DataReader **dr.** This routines copies data from the DataReader QoS properties into **qos.**

#### 2.3.2.13 ReturnCode_t get_requested_deadline_missed_status ( out RequestedDeadlineMissedStatus *status* ) [inline]

Provides access to the current RequestedDeadlineMissedStatus of the DataReader. As a side-effect, this routine will reset the **total_count_change** status field to zero.

#### 2.3.2.14 ReturnCode_t get_requested_incompatible_qos_status ( out RequestedIncompatibleQosStatus *status* ) [inline]

Provides access to the current RequestedIncompatibleQosStatus of the DataReader. As a side-effect, this routine will reset the **total_count_change** status field to zero.

#### 2.3.2.15 ReturnCode_t get_sample_lost_status ( out SampleLostStatus *status* ) [inline]

Provides access to the current SampleLostStatus of the DataReader. As a side-effect, this routine will reset the **total_count_change** status field to zero.

#### 2.3.2.16 ReturnCode_t get_sample_rejected_status ( out SampleRejectedStatus *status* ) [inline]

Provides access to the current SampleRejectedStatus of the DataReader. As a side-effect, this routine will reset the **total_count_change** status field to zero.

#### 2.3.2.17 Subscriber get_subscriber ( ) [inline]

Returns the Subscribier that contains DataReader **dr.**

#### 2.3.2.18 ReturnCode_t get_subscription_matched_status ( out SubscriptionMatchedStatus *status* ) [inline]

Provides access to the current SubscriptionMatchedStatus of the DataReader. As a side-effect, this routine will reset the **total_count_change** status field to zero.

#### 2.3.2.19 TopicDescription get_topicdescription ( ) [inline]

Returns the TopicDescription associated with DataReader **dr.**

#### 2.3.2.20 ReturnCode_t set_listener ( DataReaderListener *new_listener,* uint *mask* ) [inline]

Installs a DataReaderListener on DataReader **dr.** Only one listener may be attached to a DataReader at a time. A call to **set_listener()** will replace any current listener with **a_listener.**

**a_listener** can be NULL, which indicates a listener that does nothing.

#### 2.3.2.21 ReturnCode_t set_qos ( DataReaderQos *qos* ) [inline]

Sets the DataReaderQos values. These QoS values affect the behavior of the DataReader. This routine may fail if the provided **qos** argument is not internally consistent. In this case, DDS.RETCODE_INCONSISTENT_POLICY will be returned, and no changes will be made to the DataReader QoS.

#### 2.3.2.22 ReturnCode_t wait_for_historical_data ( Duration_t *max_wait* ) [inline]

This routine blocks until all 'historical' data is received.

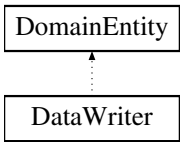
**Parameters**

| | |
|---|---|
| *max_wait* | The maximum amount of time to block while waiting. Can be set to { DDS.DURATION_INFINITE_SEC, DDS.DURATION_INFINITE_NSEC } |

## 2.4 DataWriter Class Reference

### 2.4.1 Description

The DataWriter entity provides an interface for the application to publish (write) data. The DataWriter is an abstract class that is extended to support a particular data type required by the application. A DataReader is associated with, and writes on, a single Topic. Inheritance diagram for DataWriter:




### Public Member Functions

- override ReturnCode_t enable ()
- override InstanceHandle_t get_instance_handle ()
- ReturnCode_t set_qos (DataWriterQos qos)
- ReturnCode_t get_qos (DataWriterQos qos)
- ReturnCode_t set_listener (DataWriterListener new_listener, uint mask)
- DataWriterListener get_listener ()
- Topic get_topic ()
- Publisher get_publisher ()
- ReturnCode_t wait_for_acknowledgments (Duration_t max_wait)
- ReturnCode_t assert_liveliness ()
- ReturnCode_t get_liveliness_lost_status (out LivelinessLostStatus status)
- ReturnCode_t get_offered_deadline_missed_status (out OfferedDeadlineMissedStatus status)
- ReturnCode_t get_offered_incompatible_qos_status (out OfferedIncompatibleQosStatus status)
- ReturnCode_t get_publication_matched_status (out PublicationMatchedStatus status)
- ReturnCode_t get_matched_subscriptions (List< InstanceHandle_t > subscription_handles)
- ReturnCode_t get_matched_subscription_data (SubscriptionBuiltinTopicData subscription_data, InstanceHandle_t subscription_handle)
- ReturnCode_t get_guid (GUID_t g)

### 2.4.2 Member Function Documentation

#### 2.4.2.1 ReturnCode_t assert_liveliness ( ) [inline]

This operation manually asserts the liveliness of the DataWriter **dw.** This operation is useful if the LIVELINESS QoS setting is LivelinessQosPolicyKind.MANUAL_BY_PARTICIPANT_LIVELINESS_QOS or LivelinessQosPolicyKind.MANUAL_BY_TOPIC_LIVELINESS_QOS; otherwise, it has no effect.

The **write** operation automatically asserts livelines on the DataWriter and its DomainParticipant. Therefore, **assert_liveliness** is required only if hte application is not writing data frequently enough to satisfy the LIVELINESS setting.

#### 2.4.2.2 override ReturnCode_t enable ( ) [inline], [virtual]

Enables the DataWriter. A DataWriter is created either enabled or not based on the PublisherQos setting **entity_factory.** When a DataWriter is not enabled, only the following sub-set of all DataWriter operations are legal:

- operations to get and set QoS policies,
- get_statuscondition(),
- get_status_changes(),

Any other operation may return the DDS.RETCODE_NOT_ENABLED error. DataWriter_enable() may be called on an already enabled DataWriter [it will have no effect].

Reimplemented from Entity.

#### 2.4.2.3 ReturnCode_t get_guid ( GUID_t *g* ) [inline]

Access the GUID which uniquely identifies this writer.

#### 2.4.2.4 override InstanceHandle_t get_instance_handle ( ) [inline], [virtual]

Gets the handle that locally identifies this Entity.

Reimplemented from Entity.

#### 2.4.2.5 DataWriterListener get_listener ( ) [inline]

This operation returns the currently installed DataWriterListener.

#### 2.4.2.6 ReturnCode_t get_liveliness_lost_status ( out LivelinessLostStatus *status* ) [inline]

Provides access to the current LivelinessLostStatus of the DataWriter. As a side-effect, this routine will reset the **total_count_change** status field to zero.

#### 2.4.2.7 ReturnCode_t get_matched_subscription_data ( SubscriptionBuiltinTopicData *subscription_data,* InstanceHandle_t *subscription_handle* ) [inline]

This operation returns data that describes a particular matched DataReader identified by **subscription_handle.** An appropriate handle can be obtained through a call to DataWriter.get_matched_subscriptions().

If **subscription_handle** does not identify a matched DataReader, this routine will return DDS.RETCODE_PRECONDITION_NOT_MET.

#### 2.4.2.8 ReturnCode_t get_matched_subscriptions ( List< InstanceHandle_t > *subscription_handles* ) [inline]

This operation retrieves the list of DataReaders currently matched with the DataWriter **dw.** This list will include the handles that identify DataReaders which have matching Topic and compatible QoS with DataWriter.

If a DataReader has been ignored by a call to DomainParticipant.ignore_subscription(), then it will not appear in the list.

**Parameters**

| | |
|---|---|
| *subscription_handles* | A vector that will be populated with InstanceHandle_t(s). |

#### 2.4.2.9 ReturnCode_t get_offered_deadline_missed_status ( out OfferedDeadlineMissedStatus *status* ) [inline]

Provides access to the current OfferedDeadlineMissedStatus of the DataWriter. As a side-effect, this routine will reset the **total_count_change** status field to zero.

#### 2.4.2.10 ReturnCode_t get_offered_incompatible_qos_status ( out OfferedIncompatibleQosStatus *status* ) [inline]

Provides access to the current OfferedIncompatibleQosStatus of the DataWriter. As a side-effect, this routine will reset the **total_count_change** status field to zero.

#### 2.4.2.11 ReturnCode_t get_publication_matched_status ( out PublicationMatchedStatus *status* ) [inline]

Provides access to the current PublicationMatchedStatus of the DataWriter. As a side-effect, this routine will reset the **total_count_change** and **current_count_change** status fields to zero.

#### 2.4.2.12 Publisher get_publisher ( ) [inline]

Returns the Publisher that contains DataWriter **dw.**

#### 2.4.2.13 ReturnCode_t get_qos ( DataWriterQos *qos* ) [inline]

Returns the current DataWriterQos settings held in the DataWriter **dw.** This routines copies data from the DataWriter QoS properties into **qos.**

#### 2.4.2.14 Topic get_topic ( ) [inline]

Returns the Topic associated with DataWriter **dw.**

#### 2.4.2.15 ReturnCode_t set_listener ( DataWriterListener *new_listener,* uint *mask* ) [inline]

Installs a DataWriterListener on DataWriter **dw.** Only one listener may be attached to a DataWriter at a time. A call to **set_listener()** will replace any current listener with **a_listener.**

**a_listener** can be NULL, which indicates a listener that does nothing.

#### 2.4.2.16 ReturnCode_t set_qos ( DataWriterQos *qos* ) [inline]

Sets the DataWriterQos values. These QoS values affect the behavior of the DataWriter. This routine may fail if the provided **qos** argument is not internally consistent. In this case, DDS.RETCODE_INCONSISTENT_POLICY will be returned, and no changes will be made to the DataWriter QoS.

#### 2.4.2.17 ReturnCode_t wait_for_acknowledgments ( Duration_t *max_wait* ) [inline]

Block until this writer has received acknowledgements for all written data.

This routine will block until all data written by the writer has been acknowledged, or until the 'max_wait' duration has passed. 'max_wait' can be set to INFINITE, in which case this routine may block indefinitely.

**Return values**

| | |
|---|---|
| *DDS.RETCODE_TIMEOUT* | returned if 'max_wait' passes before all acks are received |
| *DDS.RETCODE_OK* | returned if all acks have been received before 'max_wait' |

## 2.5 DDS Class Reference

### 2.5.1 Description

The 'DDS' class includes several convient constants.

#### Public Attributes

- const int LENGTH_UNLIMITED = -1
- const int DURATION_INFINITE_SEC = 0x7fffffff
- const uint DURATION_INFINITE_NSEC = 0xffffffff
- const int DURATION_ZERO_SEC = 0
- const int DURATION_ZERO_NSEC = 0
- const int TIMESTAMP_INVALID_SEC = -1
- const uint TIMESTAMP_INVALID_NSEC = 0xffffffff
- const uint READ_SAMPLE_STATE = 0x0001
- const uint NOT_READ_SAMPLE_STATE = 0x0002
- const uint ANY_SAMPLE_STATE = 0x00FF
- const uint NEW_VIEW_STATE = 0x0001
- const uint NOT_NEW_VIEW_STATE = 0x0002
- const uint ANY_VIEW_STATE = 0x00FF
- const uint ALIVE_INSTANCE_STATE = 0x0001
- const uint NOT_ALIVE_DISPOSED_INSTANCE_STATE = 0x0002
- const uint NOT_ALIVE_NO_WRITERS_INSTANCE_STATE = 0x0004
- const uint NOT_ALIVE_INSTANCE_STATE = 0x0006
- const uint ANY_INSTANCE_STATE = 0x00FF
- const uint INCONSISTENT_TOPIC_STATUS = 0x0001
- const uint OFFERED_DEADLINE_MISSED_STATUS = 0x0002
- const uint REQUESTED_DEADLINE_MISSED_STATUS = 0x0004
- const uint OFFERED_INCOMPATIBLE_QOS_STATUS = 0x0008
- const uint REQUESTED_INCOMPATIBLE_QOS_STATUS = 0x0010
- const uint SAMPLE_LOST_STATUS = 0x0020
- const uint SAMPLE_REJECTED_STATUS = 0x0040
- const uint DATA_ON_READERS_STATUS = 0x0080
- const uint DATA_AVAILABLE_STATUS = 0x0100
- const uint LIVELINESS_LOST_STATUS = 0x0200
- const uint LIVELINESS_CHANGED_STATUS = 0x0400
- const uint PUBLICATION_MATCHED_STATUS = 0x0800
- const uint SUBSCRIPTION_MATCHED_STATUS = 0x1000
- const uint ALL_STATUS = 0xffff
- const QosPolicyId_t USERDATA_QOS_POLICY_ID = QosPolicyId_t.USERDATA_QOS_POLICY_ID
- const QosPolicyId_t DURABILITY_QOS_POLICY_ID = QosPolicyId_t.DURABILITY_QOS_POLICY_ID
- const QosPolicyId_t PRESENTATION_QOS_POLICY_ID = QosPolicyId_t.PRESENTATION_QOS_POLICY_ID
- const QosPolicyId_t DEADLINE_QOS_POLICY_ID = QosPolicyId_t.DEADLINE_QOS_POLICY_ID
- const QosPolicyId_t LATENCYBUDGET_QOS_POLICY_ID = QosPolicyId_t.LATENCYBUDGET_QOS_POLICY_ID
- const QosPolicyId_t OWNERSHIP_QOS_POLICY_ID = QosPolicyId_t.OWNERSHIP_QOS_POLICY_ID
- const QosPolicyId_t OWNERSHIPSTRENGTH_QOS_POLICY_ID = QosPolicyId_t.OWNERSHIPSTRENGTH_QOS_POLICY_ID
- const QosPolicyId_t LIVELINESS_QOS_POLICY_ID = QosPolicyId_t.LIVELINESS_QOS_POLICY_ID
- const QosPolicyId_t TIMEBASEDFILTER_QOS_POLICY_ID = QosPolicyId_t.TIMEBASEDFILTER_QOS_POLICY_ID
- const QosPolicyId_t PARTITION_QOS_POLICY_ID = QosPolicyId_t.PARTITION_QOS_POLICY_ID

- const QosPolicyId_t RELIABILITY_QOS_POLICY_ID = QosPolicyId_t.RELIABILITY_QOS_POLICY_ID
- const QosPolicyId_t DESTINATIONORDER_QOS_POLICY_ID = QosPolicyId_t.DESTINATIONORDER_QOS_POLICY_ID
- const QosPolicyId_t HISTORY_QOS_POLICY_ID = QosPolicyId_t.HISTORY_QOS_POLICY_ID
- const QosPolicyId_t RESOURCELIMITS_QOS_POLICY_ID = QosPolicyId_t.RESOURCELIMITS_QOS_POLICY_ID
- const QosPolicyId_t ENTITYFACTORY_QOS_POLICY_ID = QosPolicyId_t.ENTITYFACTORY_QOS_POLICY_ID
- const QosPolicyId_t WRITERDATALIFECYCLE_QOS_POLICY_ID = QosPolicyId_t.WRITERDATALIFECYCLE_QOS_POLICY_ID
- const QosPolicyId_t READERDATALIFECYCLE_QOS_POLICY_ID = QosPolicyId_t.READERDATALIFECYCLE_QOS_POLICY_ID
- const QosPolicyId_t TOPICDATA_QOS_POLICY_ID = QosPolicyId_t.TOPICDATA_QOS_POLICY_ID
- const QosPolicyId_t GROUPDATA_QOS_POLICY_ID = QosPolicyId_t.GROUPDATA_QOS_POLICY_ID
- const QosPolicyId_t TRANSPORTPRIORITY_QOS_POLICY_ID = QosPolicyId_t.TRANSPORTPRIORITY_QOS_POLICY_ID
- const QosPolicyId_t LIFESPAN_QOS_POLICY_ID = QosPolicyId_t.LIFESPAN_QOS_POLICY_ID
- const QosPolicyId_t DURABILITYSERVICE_QOS_POLICY_ID = QosPolicyId_t.DURABILITYSERVICE_QOS_POLICY_ID
- const QosPolicyId_t DATA_REPRESENTATION_QOS_POLICY_ID = QosPolicyId_t.DATA_REPRESENTATION_QOS_POLICY_ID
- const ReturnCode_t RETCODE_OK = ReturnCode_t.RETCODE_OK
- const ReturnCode_t RETCODE_ERROR = ReturnCode_t.RETCODE_ERROR
- const ReturnCode_t RETCODE_UNSUPPORTED = ReturnCode_t.RETCODE_UNSUPPORTED
- const ReturnCode_t RETCODE_BAD_PARAMETER = ReturnCode_t.RETCODE_BAD_PARAMETER
- const ReturnCode_t RETCODE_PRECONDITION_NOT_MET = ReturnCode_t.RETCODE_PRECONDITION_NOT_MET
- const ReturnCode_t RETCODE_OUT_OF_RESOURCES = ReturnCode_t.RETCODE_OUT_OF_RESOURCES
- const ReturnCode_t RETCODE_NOT_ENABLED = ReturnCode_t.RETCODE_NOT_ENABLED
- const ReturnCode_t RETCODE_IMMUTABLE_POLICY = ReturnCode_t.RETCODE_IMMUTABLE_POLICY
- const ReturnCode_t RETCODE_INCONSISTENT_POLICY = ReturnCode_t.RETCODE_INCONSISTENT_POLICY
- const ReturnCode_t RETCODE_ALREADY_DELETED = ReturnCode_t.RETCODE_ALREADY_DELETED
- const ReturnCode_t RETCODE_TIMEOUT = ReturnCode_t.RETCODE_TIMEOUT
- const ReturnCode_t RETCODE_NO_DATA = ReturnCode_t.RETCODE_NO_DATA
- const DomainParticipantQos PARTICIPANT_QOS_DEFAULT = null
- const TopicQos TOPIC_QOS_DEFAULT = null
- const PublisherQos PUBLISHER_QOS_DEFAULT = null
- const SubscriberQos SUBSCRIBER_QOS_DEFAULT = null
- const DataWriterQos DATAWRITER_QOS_DEFAULT = null
- const DataReaderQos DATAREADER_QOS_DEFAULT = null

### Static Public Attributes

- static readonly InstanceHandle_t HANDLE_NIL = new InstanceHandle_t((IntPtr)0)

## 2.5.2 Member Data Documentation

#### 2.5.2.1 const uint ALIVE_INSTANCE_STATE = 0x0001

instance state: alive

#### 2.5.2.2 const uint ALL_STATUS = 0xffff

status flag

#### 2.5.2.3 const uint ANY_INSTANCE_STATE = 0x00FF

instance state: any

#### 2.5.2.4 const uint ANY_SAMPLE_STATE = 0x00FF

sample state: any

#### 2.5.2.5 const uint ANY_VIEW_STATE = 0x00FF

view state: any

#### 2.5.2.6 const uint DATA_AVAILABLE_STATUS = 0x0100

status flag

#### 2.5.2.7 const uint DATA_ON_READERS_STATUS = 0x0080

status flag

#### 2.5.2.8 const QosPolicyId_t DATA_REPRESENTATION_QOS_POLICY_ID = QosPolicyId_t.DATA_REPRESENTATION_QOS_POLICY_ID

policy id

#### 2.5.2.9 const DataReaderQos DATAREADER_QOS_DEFAULT = null

qos default

#### 2.5.2.10 const DataWriterQos DATAWRITER_QOS_DEFAULT = null

qos default

#### 2.5.2.11 const QosPolicyId_t DEADLINE_QOS_POLICY_ID = QosPolicyId_t.DEADLINE_QOS_POLICY_ID

policy id

#### 2.5.2.12 const QosPolicyId_t DESTINATIONORDER_QOS_POLICY_ID = QosPolicyId_t.DESTINATIONORDER_QOS_POLICY_ID

policy id

#### 2.5.2.13 const QosPolicyId_t DURABILITY_QOS_POLICY_ID = QosPolicyId_t.DURABILITY_QOS_POLICY_ID

policy id

#### 2.5.2.14 const QosPolicyId_t DURABILITYSERVICE_QOS_POLICY_ID = QosPolicyId_t.DURABILITYSERVICE_QOS_POLICY_ID

policy id

#### 2.5.2.15 const uint DURATION_INFINITE_NSEC = 0xffffffff

infinite nano seconds

#### 2.5.2.16 const int DURATION_INFINITE_SEC = 0x7fffffff

infinite seconds

#### 2.5.2.17 const int DURATION_ZERO_NSEC = 0

zero nano seconds

#### 2.5.2.18 const int DURATION_ZERO_SEC = 0

zero seconds

#### 2.5.2.19 const QosPolicyId_t ENTITYFACTORY_QOS_POLICY_ID = QosPolicyId_t.ENTITYFACTORY_QOS_POLICY_ID

policy id

#### 2.5.2.20 const QosPolicyId_t GROUPDATA_QOS_POLICY_ID = QosPolicyId_t.GROUPDATA_QOS_POLICY_ID

policy id

#### 2.5.2.21 readonly InstanceHandle_t HANDLE_NIL = new InstanceHandle_t((IntPtr)0) [static]

nil handle

#### 2.5.2.22 const QosPolicyId_t HISTORY_QOS_POLICY_ID = QosPolicyId_t.HISTORY_QOS_POLICY_ID

policy id

#### 2.5.2.23 const uint INCONSISTENT_TOPIC_STATUS = 0x0001

status flag

#### 2.5.2.24 const QosPolicyId_t LATENCYBUDGET_QOS_POLICY_ID = QosPolicyId_t.LATENCYBUDGET_QOS_POLICY_ID

policy id

#### 2.5.2.25 const int LENGTH_UNLIMITED = -1

unlimited length

#### 2.5.2.26 const QosPolicyId_t LIFESPAN_QOS_POLICY_ID = QosPolicyId_t.LIFESPAN_QOS_POLICY_ID

policy id

#### 2.5.2.27 const uint LIVELINESS_CHANGED_STATUS = 0x0400

status flag

#### 2.5.2.28 const uint LIVELINESS_LOST_STATUS = 0x0200

status flag

#### 2.5.2.29 const QosPolicyId_t LIVELINESS_QOS_POLICY_ID = QosPolicyId_t.LIVELINESS_QOS_POLICY_ID

policy id

#### 2.5.2.30 const uint NEW_VIEW_STATE = 0x0001

view state: new

#### 2.5.2.31 const uint NOT_ALIVE_DISPOSED_INSTANCE_STATE = 0x0002

instance state: disposed

#### 2.5.2.32 const uint NOT_ALIVE_INSTANCE_STATE = 0x0006

instance state: not alive (either disposed or no writers)

#### 2.5.2.33 const uint NOT_ALIVE_NO_WRITERS_INSTANCE_STATE = 0x0004

instance state: no writers

#### 2.5.2.34 const uint NOT_NEW_VIEW_STATE = 0x0002

view state: not new

#### 2.5.2.35 const uint NOT_READ_SAMPLE_STATE = 0x0002

sample state: not read

#### 2.5.2.36 const uint OFFERED_DEADLINE_MISSED_STATUS = 0x0002

status flag

#### 2.5.2.37 const uint OFFERED_INCOMPATIBLE_QOS_STATUS = 0x0008

status flag

#### 2.5.2.38 const QosPolicyId_t OWNERSHIP_QOS_POLICY_ID = QosPolicyId_t.OWNERSHIP_QOS_POLICY_ID

policy id

#### 2.5.2.39 const QosPolicyId_t OWNERSHIPSTRENGTH_QOS_POLICY_ID = QosPolicyId_t.OWNERSHIPSTRENGTH_QOS_POLICY_ID

policy id

#### 2.5.2.40 const DomainParticipantQos PARTICIPANT_QOS_DEFAULT = null

qos default

#### 2.5.2.41 const QosPolicyId_t PARTITION_QOS_POLICY_ID = QosPolicyId_t.PARTITION_QOS_POLICY_ID

policy id

#### 2.5.2.42 const QosPolicyId_t PRESENTATION_QOS_POLICY_ID = QosPolicyId_t.PRESENTATION_QOS_POLICY_ID

policy id

#### 2.5.2.43 const uint PUBLICATION_MATCHED_STATUS = 0x0800

status flag

#### 2.5.2.44 const PublisherQos PUBLISHER_QOS_DEFAULT = null

qos default

#### 2.5.2.45 const uint READ_SAMPLE_STATE = 0x0001

sample state: read

#### 2.5.2.46 const QosPolicyId_t READERDATALIFECYCLE_QOS_POLICY_ID = QosPolicyId_t.READERDATALIFECYCLE_QOS_POLICY_ID

policy id

#### 2.5.2.47 const QosPolicyId_t RELIABILITY_QOS_POLICY_ID = QosPolicyId_t.RELIABILITY_QOS_POLICY_ID

policy id

#### 2.5.2.48 const uint REQUESTED_DEADLINE_MISSED_STATUS = 0x0004

status flag

#### 2.5.2.49 const uint REQUESTED_INCOMPATIBLE_QOS_STATUS = 0x0010

status flag

#### 2.5.2.50 const QosPolicyId_t RESOURCELIMITS_QOS_POLICY_ID = QosPolicyId_t.RESOURCELIMITS_QOS_POLICY_ID

policy id

#### 2.5.2.51 const ReturnCode_t RETCODE_ALREADY_DELETED = ReturnCode_t.RETCODE_ALREADY_DELETED

return code

#### 2.5.2.52 const ReturnCode_t RETCODE_BAD_PARAMETER = ReturnCode_t.RETCODE_BAD_PARAMETER

return code

#### 2.5.2.53 const ReturnCode_t RETCODE_ERROR = ReturnCode_t.RETCODE_ERROR

return code

#### 2.5.2.54 const ReturnCode_t RETCODE_IMMUTABLE_POLICY = ReturnCode_t.RETCODE_IMMUTABLE_POLICY

return code

#### 2.5.2.55 const ReturnCode_t RETCODE_INCONSISTENT_POLICY = ReturnCode_t.RETCODE_INCONSISTENT_POLICY

return code

#### 2.5.2.56 const ReturnCode_t RETCODE_NO_DATA = ReturnCode_t.RETCODE_NO_DATA

return code

#### 2.5.2.57 const ReturnCode_t RETCODE_NOT_ENABLED = ReturnCode_t.RETCODE_NOT_ENABLED

return code

#### 2.5.2.58 const ReturnCode_t RETCODE_OK = ReturnCode_t.RETCODE_OK

return code

#### 2.5.2.59 const ReturnCode_t RETCODE_OUT_OF_RESOURCES = ReturnCode_t.RETCODE_OUT_OF_RESOURCES

return code

#### 2.5.2.60 const ReturnCode_t RETCODE_PRECONDITION_NOT_MET = ReturnCode_t.RETCODE_PRECONDITION_NOT_MET

return code

#### 2.5.2.61 const ReturnCode_t RETCODE_TIMEOUT = ReturnCode_t.RETCODE_TIMEOUT

return code

#### 2.5.2.62 const ReturnCode_t RETCODE_UNSUPPORTED = ReturnCode_t.RETCODE_UNSUPPORTED

return code

#### 2.5.2.63 const uint SAMPLE_LOST_STATUS = 0x0020

status flag

#### 2.5.2.64 const uint SAMPLE_REJECTED_STATUS = 0x0040

status flag

#### 2.5.2.65 const SubscriberQos SUBSCRIBER_QOS_DEFAULT = null

qos default

#### 2.5.2.66 const uint SUBSCRIPTION_MATCHED_STATUS = 0x1000

status flag

#### 2.5.2.67 const QosPolicyId_t TIMEBASEDFILTER_QOS_POLICY_ID = QosPolicyId_t.TIMEBASEDFILTER_QOS_POLICY_ID

policy id

#### 2.5.2.68 const uint TIMESTAMP_INVALID_NSEC = 0xffffffff

invalid nano seconds

#### 2.5.2.69 const int TIMESTAMP_INVALID_SEC = -1

invalid seconds

#### 2.5.2.70 const TopicQos TOPIC_QOS_DEFAULT = null

qos default

#### 2.5.2.71 const QosPolicyId_t TOPICDATA_QOS_POLICY_ID = QosPolicyId_t.TOPICDATA_QOS_POLICY_ID

policy id

#### 2.5.2.72 const QosPolicyId_t TRANSPORTPRIORITY_QOS_POLICY_ID = QosPolicyId_t.TRANSPORTPRIORITY_QOS_POLICY_ID

policy id

#### 2.5.2.73 const QosPolicyId_t USERDATA_QOS_POLICY_ID = QosPolicyId_t.USERDATA_QOS_POLICY_ID

policy id

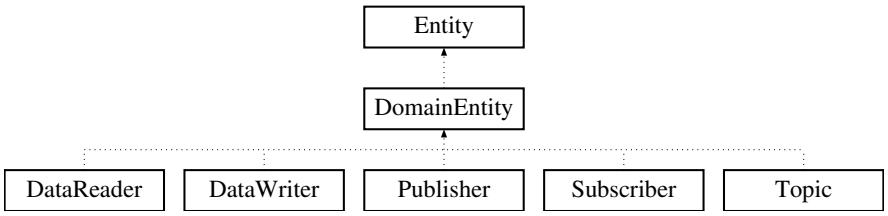
#### 2.5.2.74 const QosPolicyId_t WRITERDATALIFECYCLE_QOS_POLICY_ID = QosPolicyId_t.WRITERDATALIFECYCLE_QOS_POLICY_ID

policy id

## 2.6 DomainEntity Class Reference

### 2.6.1 Description

Base class for all DDS Domain Entities. Inheritance diagram for DomainEntity:



## 2.7 DomainParticipantFactory Class Reference

### 2.7.1 Description

DomainParticipantFactory constructs DomainParticipants.

The DomainParticipantFactory is used to configure, create and destroy DomainParticipant instances.

**Author**

Twin Oaks Computing, Inc

### Public Member Functions

- DomainParticipant create_participant (uint domain_id, DomainParticipantQos qos, DomainParticipantListener listener, uint mask)
- ReturnCode_t delete_participant (DomainParticipant participant)
- DomainParticipant lookup_participant (uint domain_id)
- ReturnCode_t set_default_participant_qos (DomainParticipantQos qos)
- ReturnCode_t get_default_participant_qos (DomainParticipantQos qos)
- ReturnCode_t set_qos (DomainParticipantFactoryQos qos)
- ReturnCode_t get_qos (DomainParticipantFactoryQos qos)
- ReturnCode_t set_license (string lic)
- ReturnCode_t set_license_debug (bool debug)

### Static Public Member Functions

- static DomainParticipantFactory get_instance ()

### Static Public Attributes

- static DomainParticipantFactory Instance

### 2.7.2 Member Function Documentation

#### 2.7.2.1 DomainParticipant create_participant ( uint *domain_id,* DomainParticipantQos *qos,* DomainParticipantListener *listener,* uint *mask* ) [inline]

Creates a DomainParticipant.

This operation creates a new DomainParticipant object. The caller provides the domain_id to which the Participant should belong. The listener and mask arguments are used to specify a set of callback routines which will be invoked upon detection of certain events. The qos argument specifies the DomainParticipant Quality of Service settings that should be used when creating the DomainParticipant. It may be specified as DDS.PARTICIPANT_QOS_DEFAULT to instruct CoreDX to use the default qos settings held in the DomainParticipantFactory. This routine will return NULL if it fails to create a DomainParticipant.

**Returns**

DomainParticipant

#### 2.7.2.2 ReturnCode_t delete_participant ( DomainParticipant *participant* ) [inline]

Destroys the provided DomainParticipant.

This routine will fail if all Entities (Publishers, Subscribers, etc) created through the specified DomainParticipant have not yet been deleted. (In this case, RETCODE_PRECONDITION_NOT_MET will be returned.)

#### 2.7.2.3 ReturnCode_t get_default_participant_qos ( DomainParticipantQos *qos* ) [inline]

Provides access to the default Participant qos held in the factory.

The provided **qos** argument is populated with the default qos settings.

#### 2.7.2.4 static DomainParticipantFactory get_instance ( ) [inline], [static]

Get access to the singleton DomainParticipantFactory.

#### 2.7.2.5 ReturnCode_t get_qos ( DomainParticipantFactoryQos *qos* ) [inline]

Provides access to the QoS settings of the DomainParticipantFactory.

#### 2.7.2.6 DomainParticipant lookup_participant ( uint *domain_id* ) [inline]

Returns a previously created DomainParticipant belonging to the specified **domain_id**.

If there are multiple DomainParticipants in existence within the specified domain, one of them will be returned.

#### 2.7.2.7 ReturnCode_t set_default_participant_qos ( DomainParticipantQos *qos* ) [inline]

Sets the default DDS.DomainParticipantQos held in the factory.

This default qos will be used during subsequent calls to DDS.DomainParticipantFactory.create_participant() if the special DDS.PARTICIPANT_QOS_DEFAULT value is provided for **qos**.
This routine may fail if the provided **qos** argument is not internally consistent. In this case, DDS.RETCODE_INCONSISTENT_POLICY will be returned, and no changes will be made to the DomainParticipantFactory.

#### 2.7.2.8 ReturnCode_t set_license ( string *lic* ) [inline]

Configures the license file or license string to enable CoreDX DDS.

CoreDX DDS normally requires a license to run. This license can be sepecified several ways:

- By environment variable (for example: TLM_LICENSE)
- By calling DomainParticipantFactory.set_license_file() with the full path and name of a license file
- By calling DomainParticipantFactory.set_license_file() with the license string enclosed in "<" ">"

#### 2.7.2.9 ReturnCode_t set_license_debug ( bool *debug* ) [inline]

Enables debug output from the CoreDX DDS run-time license system.

This routine will configure CoreDX DDS to generate debug information in processing the license at run-time. The 'deubg' parameter may be either: 'nonzero' to enable debug output or 'zero' to disable debug ouput.

#### 2.7.2.10 ReturnCode_t set_qos ( DomainParticipantFactoryQos *qos* ) [inline]

Sets the DomainParticipantFactory QoS values.

These QoS values affect the behavior of the factory.

This routine may fail if the provided **qos** argument is not internally consistent. In this case, DDS.INCONSISTENT_POLICY will be returned, and no changes will be made to the DomainParticipantFactory.

### 2.7.3 Member Data Documentation

#### 2.7.3.1 DomainParticipantFactory Instance [static]

**Initial value:**

```
{
      get
        {
          return dpf;
        }
    }

    [DllImport(DDS.COREDX_DLL, EntryPoint="DDS_DomainParticipant_set_user", CallingConvention=DDS.cconv)]
    private static extern int DDS_DomainParticipant_set_user(IntPtr dp, IntPtr user)
```

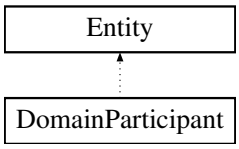
A C# style property with accessor. use it like this: DomainParticipantFactory.Instance.create_participant()

## 2.8 DomainParticipant Class Reference

### 2.8.1 Description

The DomainParticipant is used to configure, create and destroy Publisher, Subscriber and Topic objects.

The DomainParticipant is a container for the objects that it creates. The objects operate within the **domain** identified by the **domain_id** provided when the DomainParticipant was created. Objects within different **domains** do not communicate or interfere with each other. Inheritance diagram for DomainParticipant:




### Public Member Functions

- override ReturnCode_t enable ()
- override InstanceHandle_t get_instance_handle ()
- ReturnCode_t get_qos (DomainParticipantQos qos)
- ReturnCode_t set_qos (DomainParticipantQos qos)
- ReturnCode_t set_listener (DomainParticipantListener new_listener, uint mask)
- DomainParticipantListener get_listener ()
- Publisher create_publisher (PublisherQos qos, PublisherListener listener, uint mask)
- ReturnCode_t delete_publisher (Publisher p)
- Subscriber create_subscriber (SubscriberQos qos, SubscriberListener listener, uint mask)
- ReturnCode_t delete_subscriber (Subscriber s)
- Topic create_topic (string topic_name, string type_name, TopicQos qos, TopicListener listener, uint mask)
- ReturnCode_t delete_topic (Topic topic)
- ContentFilteredTopic create_contentfilteredtopic (String name, Topic related_topic, String filter_expression, List< String > filter_parameters)
- ReturnCode_t delete_contentfilteredtopic (ContentFilteredTopic cft)
- MultiTopic create_multitopic (String name, String type_name, String subscription_expression, List< String > expression_params)
- ReturnCode_t delete_multitopic (MultiTopic a_multitopic)
- Topic find_topic (String topic_name, Duration_t timeout)
- TopicDescription lookup_topicdescription (String name)
- Subscriber get_builtin_subscriber ()
- ReturnCode_t ignore_participant (InstanceHandle_t handle)
- ReturnCode_t ignore_topic (InstanceHandle_t handle)
- ReturnCode_t ignore_publication (InstanceHandle_t handle)
- ReturnCode_t ignore_subscription (InstanceHandle_t handle)
- long get_domain_id ()
- ReturnCode_t delete_contained_entities ()
- ReturnCode_t assert_liveliness ()
- ReturnCode_t set_default_publisher_qos (PublisherQos qos)
- ReturnCode_t get_default_publisher_qos (PublisherQos qos)
- ReturnCode_t set_default_subscriber_qos (SubscriberQos qos)
- ReturnCode_t get_default_subscriber_qos (SubscriberQos qos)

- ReturnCode_t set_default_topic_qos (TopicQos qos)
- ReturnCode_t get_default_topic_qos (TopicQos qos)
- ReturnCode_t get_discovered_participants (List< InstanceHandle_t > participant_handles)
- ReturnCode_t get_discovered_participant_data (ParticipantBuiltinTopicData participant_data, InstanceHandle_t participant_handle)
- ReturnCode_t get_discovered_topics (List< InstanceHandle_t > topic_handles)
- ReturnCode_t get_discovered_topic_data (TopicBuiltinTopicData topic_data, InstanceHandle_t topic_handle)
- bool contains_entity (InstanceHandle_t handle)
- ReturnCode_t get_current_time (ref Time_t current_time)
- ReturnCode_t register_type (TypeSupport ts, string type_name)
- ReturnCode_t add_transport (Transport t)
- ReturnCode_t do_work (Duration_t time_slice)
- ReturnCode_t builtin_wait_for_acknowledgments (Duration_t max_wait)
- void print_stats ()
- ReturnCode_t get_guid (GUID_t g)

### 2.8.2 Member Function Documentation

#### 2.8.2.1 ReturnCode_t add_transport ( Transport *t* ) [inline]

Install the provided Transport.

Must be called before enabling the DomainParticipant. The DomainParticipant will default to using standard UDP RTPS transport, if no other transports are added.

**See also**

LmtTransportConfig
LmtTransport
TcpTransportConfig
TcpTransport
UdpTransportConfig
UdpTransport

#### 2.8.2.2 ReturnCode_t assert_liveliness ( ) [inline]

This operation manually asserts the liveliness of the DomainParticipant **dp.** This operation indicates that the DomainParticipant is still alive. It is effective only if the DomainParticipant contains on ore more DataWriter objects with LIVELINES QoS set to MANUAL_BY_PARTICIPANT_LIVELINESS_QOS. In this case, the operation will assert the liveliness of those particular DataWriter objects.

Writing data via the DataWriter **write** operation automatically asserts the liveliness of the DataWriter and its containing DomainParticipant. Therefore, the use of DomainParticipant.assert_liveliness() is needed only if the application is not writing data frequently enough to keep the DataWriter considered 'alive'.

#### 2.8.2.3 ReturnCode_t builtin_wait_for_acknowledgments ( Duration_t *max_wait* ) [inline]

Ensure that local discovery information has been received by all known peers.

#### 2.8.2.4 bool contains_entity ( InstanceHandle_t *handle* ) [inline]

This operation checks whether or not the given handle **a_handle** represents an object that was created by the DomainParticipant **d.** This applies recursively to all objects created by the DomainParticipant.

The routine will return true (non-zero) if the handle is found, zero otherwise.

#### 2.8.2.5 ContentFilteredTopic create_contentfilteredtopic ( String *name,* Topic *related_topic,* String *filter_expression,* List< String > *filter_parameters* ) [inline]

This operation creates a ContentFilteredTopic.

The ContentFilteredTopic is associated with another un-filtered topic **related_topic.**

The **filter_expression** is an SQL like condition expression, and **filter_parameters** provide optional parameters that are referenced by the **filter_expression.** The syntax of the filter expression is similar to the WHERE clause in SQL. For example "x<4" is a valid filter expression. It would test that data member 'x' is less than value '4'. If the filter expression evaluates to TRUE, then the data sample will be available, otherwise the data sample would be 'filtered' (excluded).

The filter_expression can refer to parameters. The syntax for paramters is the percent sign '' followed by a number. The number is the index of the paramter in the **filter_paramters** sequence. Parameters are counted starting at zero. So, "%0" refers to the first parameter, and "%4" refers to the fifth paramter. Using this syntax, the expression "x<%0" would test the value of 'x' against the first parameter in the sequence.

#### 2.8.2.6 MultiTopic create_multitopic ( String *name,* String *type_name,* String *subscription_expression,* List< String > *expression_params* ) [inline]

**Not Yet Supported** This is currently unsupported in the C# language binding.

#### 2.8.2.7 Publisher create_publisher ( PublisherQos *qos,* PublisherListener *listener,* uint *mask* ) [inline]

This operation creates a Publisher with the specified PublisherQoS settings and PublisherListener. The value **DDS.PUBLISHER_QOS_DEFAULT** may be provided for the **qos** argument. This will indicate that the the default publisher QoS settings held in the DomainParticipant should be used.

This operation may fail and return NULL if the QoS settings are internally inconsistent.

#### 2.8.2.8 Subscriber create_subscriber ( SubscriberQos *qos,* SubscriberListener *listener,* uint *mask* ) [inline]

This operation creates a Subscriber with the specified SubscriberQos and SubscriberListener. The value **DDS.SUBSCRIBER_QOS_DEFAULT** may be provided for the **qos** argument. This will indicate that the the default subscriber QoS settings held in the DomainParticipant should be used.

This operation may fail and return NULL if the QoS settings are internally inconsistent.

#### 2.8.2.9 Topic create_topic ( string *topic_name,* string *type_name,* TopicQos *qos,* TopicListener *listener,* uint *mask* ) [inline]

This operation creates a Topic with the specified TopicQos settings and TopicListener.

The value **DDS.TOPIC_QOS_DEFAULT** may be provided for the **qos** argument. This will indicate that the the default topic QoS settings held in the DomainParticipant should be used.

The topic is created with a reference to the data type indicated by the **type_name** argument. The data type must have been registered with the DomainParticipant (by a call to **register_type()** on the appropriate **TypeSupport** object) prior to calling **create_topic().**

This operation may fail and return NULL if the QoS settings are internally inconsistent.

The topic returned from this operation (if not NULL) must be deleted by calling DomainParticipant.delete_topic().

#### 2.8.2.10 ReturnCode_t delete_contained_entities ( ) [inline]

This operation deletes all the objects created by means of the **create** operations on DomainParticipant **dp.** This routine will recursively call the corresponding delete_contained_entities() operation on each of the contained objects (Publishers, Subscribers, Topics, ContentFilteredTopics, and MultiTopics). If successful, this operation will recursively delete all objects contained with this DomainParticipant. After successful execution, the application may delete the DomainParticipant by calling DomainParticipantFactory.delete_participant().

If any of the objects cannot be deleted, this routine will return ReturnCode_t.RETCODE_PRECONDITION_NOT_MET.

#### 2.8.2.11 ReturnCode_t delete_contentfilteredtopic ( ContentFilteredTopic *cft* ) [inline]

This operation deletes a previously allocated ContentFilteredTopic. This operation will fail if there are any DataReader, DataWriter, ContentFilteredTopic, or MultiTopic objects that reference the specified Topic. In this case, the operation will return ReturnCode_t.RETCODE_PRECONDITION_NOT_MET.

The Topic must be owned by DomainPartipant **dp;** otherwise ReturnCode_t.RETCODE_PRECONDITION_NOT_MET will be returned.

#### 2.8.2.12 ReturnCode_t delete_multitopic ( MultiTopic *a_multitopic* ) [inline]

**Not Yet Supported** This is currently unsupported in the C# language binding.

#### 2.8.2.13 ReturnCode_t delete_publisher ( Publisher *p* ) [inline]

This operation deletes an existing Publisher. A Publisher cannot be deleted if it contains any DataWriter objects. In this case, **delete_publisher** will return ReturnCode_t.RETCODE_PRECONDITION_NOT_MET.

If the DomainParticipant **dp** does not contain the provided Publisher **p,** the operation will return **ReturnCode_t.RETCODE_PRECONDITION_NOT_MET.**

#### 2.8.2.14 ReturnCode_t delete_subscriber ( Subscriber *s* ) [inline]

This operation deletes an existing Subscriber. A Subscriber cannot be deleted if it contains any DataReader objects. In this case, **delete_subscriber** will return ReturnCode_t.RETCODE_PRECONDITION_NOT_MET.

If the DomainParticipant **dp** does not contain the provided Subscriber **s,** the operation will return **ReturnCode_t.RETCODE_PRECONDITION_NOT_MET.**

#### 2.8.2.15 ReturnCode_t delete_topic ( Topic *topic* ) [inline]

This operation deletes a Topic. This operation will fail if there are any DataReader, DataWriter, ContentFilteredTopic, or MultiTopic objects that reference the specified Topic. In this case, the operation will return ReturnCode_t.RETCODE_PRECONDITION_NOT_MET.

The Topic must be owned by DomainPartipant **dp;** otherwise ReturnCode_t.RETCODE_PRECONDITION_NOT_MET will be returned.

#### 2.8.2.16 ReturnCode_t do_work ( Duration_t *time_slice* ) [inline]

Transfer control to the DomainParticipant so it can do background processing.

This function is to be called only if the DomainParticipant is configured to use NO threads via the thread_model QoS policy. The participant will continue to execute background processing until at least 'time_slice' time passes. This routine will attempt to use no more than 'time_slice' time, but this is not a guarantee.

**Note**

If the DomainParticipant has threads enabled (default case), then the Participant takes care of scheduling background work on its own, and this routine is not necessary and should not be called.

#### 2.8.2.17 override ReturnCode_t enable ( ) [inline], [virtual]

Enables the DomainParticipant.

A DomainParticipant is created either enabled or not based on the DomainParticipantFactoryQoS setting **entity_factory.** When a DomainPartcipant is not enabled, only the following sub-set of all DomainParticipant operations are legal:

- operations to get and set QoS policies,
- factory operations (create, delete),
- get_statuscondition(),
- get_status_changes(),
- lookup operations

Any other DomainParticipant operation will return the ReturnCode_t.RETCODE_NOT_ENABLED error. DomainParticipant.enable() may be called on an already enabled DomainParticipant [it will have no effect].

Reimplemented from Entity.

#### 2.8.2.18 Topic find_topic ( String *topic_name,* Duration_t *timeout* ) [inline]

This operation returns a Topic identified by the provided **topic_name.** If a topic with name **topic_name** is known to exist, the function returns this matching topic immediately. Otherwise, the operation will block for up to the **timeout** duration. If a topic with name **topic_name** is discovered or otherwise created, the operation will cease to wait, and will return the matching topic.

If a matching topic is not known by the time the **timout** has expired, the operation will return NULL.

Like DomainParticipant_create_topic(), the topic returned from this operation (if not NULL) must be deleted by calling DomainParticipant.delete_topic().

#### 2.8.2.19 Subscriber get_builtin_subscriber ( ) [inline]

Returns the built-in Subscriber.

Each DomainParticipant contains several built-in Topic objects as well as corresponding DataReader objects to access them. These built-in DataReader objects belong to the single built-in Subscriber that is accessed through this operation.

The built-in Topics are used to communicate information about other DomainParticipant, Topic, DataReader, and DataWriter objects.

An application should not explicitly delete the Subscriber returned by this operation - it is managed internally.

**See also**

ParticipantBuiltinTopicDataDataReader
PublicationBuiltinTopicDataDataReader
SubscriptionBuiltinTopicDataDataReader

#### 2.8.2.20 ReturnCode_t get_current_time ( ref Time_t *current_time* ) [inline]

This operation returns the current value of the time used by the service.

#### 2.8.2.21 ReturnCode_t get_default_publisher_qos ( PublisherQos *qos* ) [inline]

Provides access to the default PublisherQos settings held in the factory. The provided **qos** argument is populated with the default qos settings.

#### 2.8.2.22 ReturnCode_t get_default_subscriber_qos ( SubscriberQos *qos* ) [inline]

Provides access to the default SubscriberQos settings held in the factory. The provided **qos** argument is populated with the default qos settings.

#### 2.8.2.23 ReturnCode_t get_default_topic_qos ( TopicQos *qos* ) [inline]

Provides access to the default TopicQos settings held in the factory. The provided **qos** argument is populated with the default qos settings.

#### 2.8.2.24 ReturnCode_t get_discovered_participant_data ( ParticipantBuiltinTopicData *participant_data,* InstanceHandle_t *participant_handle* ) [inline]

This operation returns data that describes a particular discovered participant identified by **participant_handle.** An appropriate handle can be obtained through a call to DomainParticipant.get_discovered_participants().

If **participant_handle** does not identify a known DomainParticipant, this routine will return ReturnCode_t.RETCODE_PRECONDITION_N

#### 2.8.2.25 ReturnCode_t get_discovered_participants ( List< InstanceHandle_t > *participant_handles* ) [inline]

This operation returns the list of handles identifying the DomainParticipant objects that have been discovered. The returned list will include only participants which are in the same domain as participant **d,** and which are not explicitly ignored as a result of a call to DomainParticipant.ignore_participant().

#### 2.8.2.26 ReturnCode_t get_discovered_topic_data ( TopicBuiltinTopicData *topic_data,* InstanceHandle_t *topic_handle* ) [inline]

This operation returns data that describes a particular discovered topic identified by **topic_handle.** An appropriate handle can be obtained through a call to DomainParticipant.get_discovered_topics().

If **topic_handle** does not identify a known Topic, this routine will return ReturnCode_t.RETCODE_PRECONDITION_NOT_MET.

**Not Yet Supported** This is currently unsupported in the C# language binding.

#### 2.8.2.27 ReturnCode_t get_discovered_topics ( List< InstanceHandle_t > *topic_handles* ) [inline]

This operation returns the list of handles identifying the Topic objects that have been discovered. The returned list will include only topics which are in the same domain as participant **d,** and which are not explicitly ignored as a result of a call to DomainParticipant.ignore_topic().

**Not Yet Supported** This is currently unsupported in the C# language binding.

#### 2.8.2.28 long get_domain_id ( ) [inline]

Gets the **domain_id** to which the DomainParticipant belongs.

#### 2.8.2.29 ReturnCode_t get_guid ( GUID_t *g* ) [inline]

Access the GUID which uniquely identifies this participant.

#### 2.8.2.30 override InstanceHandle_t get_instance_handle ( ) [inline], [virtual]

Gets the handle that locally identifies this Entity.

Reimplemented from Entity.

#### 2.8.2.31 DomainParticipantListener get_listener ( ) [inline]

This operation returns the currently installed DomainParticipantListener. Because the infrastructure makes a copy of the listener provided in DomainParticipant.set_listener(), the returned structure pointer will not match the pointer originally provided. However, the function pointers within the structure will match. Also, the application should not free the data referenced by the returned pointer.

#### 2.8.2.32 ReturnCode_t get_qos ( DomainParticipantQos *qos* ) [inline]

Gets the DomainParticipantQoS settings of the DomainParticipant.

#### 2.8.2.33 ReturnCode_t ignore_participant ( InstanceHandle_t *handle* ) [inline]

Instructs the DomainParticipant **dp** to ignore the external DomainParticipant identified by **handle.** This will cause the infrastructure to behave as if the external participant **handle** did not exist. The handle can be discovered by accessing the built-in topic **DCPSParticipant** via the appropriate built-in DataReader.

There is no mechanism to reverse this operation.

#### 2.8.2.34 ReturnCode_t ignore_publication ( InstanceHandle_t *handle* ) [inline]

This operation instructs the DomainParticipant **dp** to ignore a Publication identified by **handle.** The handle can be discovered by accessing the built-in topic **DCPSPublication** via the appropriate built-in DataReader.

There is no mechanism to reverse this operation.

**Not Yet Supported** This is currently unsupported in the C# language binding.

#### 2.8.2.35 ReturnCode_t ignore_subscription ( InstanceHandle_t *handle* ) [inline]

This operation instructs the DomainParticipant **dp** to ignore a Subscription identified by **handle.** The handle can be discovered by accessing the built-in topic **DCPSSubscription** via the appropriate built-in DataReader.

There is no mechanism to reverse this operation.

**Not Yet Supported** This is currently unsupported in the C# language binding.

#### 2.8.2.36 ReturnCode_t ignore_topic ( InstanceHandle_t *handle* ) [inline]

This operation instructs the DomainParticipant **dp** to ignore a Topic identified by **handle.** This can be used to save resources if a participant will never participate on certain Topics. The handle can be discovered by accessing the built-in topic **DCPSTopic** via the appropriate built-in DataReader.

There is no mechanism to reverse this operation.

**Not Yet Supported** This is currently unsupported in the C# language binding.

#### 2.8.2.37 TopicDescription lookup_topicdescription ( String *name* ) [inline]

This operation returns an existing, locally-created TopicDescription, named **name.** If a TopicDescription named **name** does not exist, then this routine will return NULL.

#### 2.8.2.38 void print_stats ( ) [inline]

Send stats on Reader/Writer data caches.

#### 2.8.2.39 ReturnCode_t register_type ( TypeSupport *ts,* string *type_name* ) [inline]

Registers a TypeSupport with the Participant.

Associates a TypeSupport object with the provided 'type_name'. This TypeSupport will be used to handle data that has a matching type_name. If a TypeSupport has already been registered for the provided type_name, and the new type does not match the previously registered type, then an error is returned and the previously registered TypeSupport is maintained. [Upon error, the new type is not registered.]

#### 2.8.2.40 ReturnCode_t set_default_publisher_qos ( PublisherQos *qos* ) [inline]

Sets the default PublisherQos held in the DomainParticipant. This default qos will be used during subsequent calls to DomainParticipant.create_publisher() if the special DDS.PUBLISHER_QOS_DEFAULT value is provided for

**qos**. This routine may fail if the provided **qos** argument is not internally consistent. In this case, ReturnCode_t.RETCODE_INCONSISTENT_POLICY will be returned, and no changes will be made to the DomainParticipant.

#### 2.8.2.41 ReturnCode_t set_default_subscriber_qos ( SubscriberQos *qos* ) [inline]

Sets the default SubscriberQos held in the DomainParticipant. This default qos will be used during subsequent calls to DomainParticipant.create_subscriber() if the special DDS.SUBSCRIBER_QOS_DEFAULT value is provided for **qos**. This routine may fail if the provided **qos** argument is not internally consistent. In this case, ReturnCode_t.RETCODE_INCONSISTENT_POLICY will be returned, and no changes will be made to the DomainParticipant.

#### 2.8.2.42 ReturnCode_t set_default_topic_qos ( TopicQos *qos* ) [inline]

Sets the default TopicQos held in the DomainParticipant. This default qos will be used during subsequent calls to DomainParticipant.create_topic() [and related] if the special DDS.TOPIC_QOS_DEFAULT value is provided for **qos**. This routine may fail if the provided **qos** argument is not internally consistent. In this case, ReturnCode_t.RETCODE_INCONSISTENT_POLICY will be returned, and no changes will be made to the DomainParticipant.

#### 2.8.2.43 ReturnCode_t set_listener ( DomainParticipantListener *new_listener,* uint *mask* ) [inline]

This operation installs a DomainParticipantListener on the DomainParticipant. Only one listener may be attached to a DomainParticipant at a time. A call to **set_listener()** will replace any current listener with **a_listener.**

**a_listener** can be NULL, which indicates a listener that does nothing.

The infrastructure will make an internal copy of the listener structure so that it need not be persisted by the application.

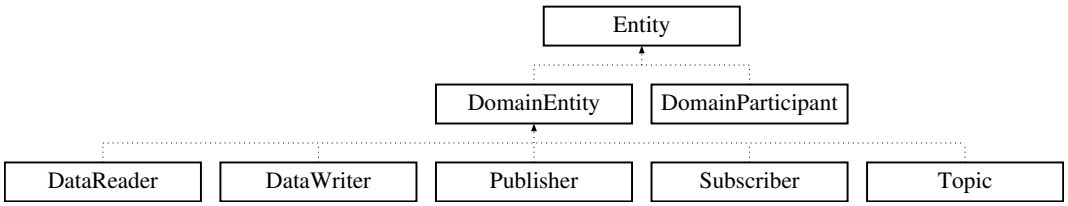
#### 2.8.2.44 ReturnCode_t set_qos ( DomainParticipantQos *qos* ) [inline]

Sets the DomainParticipantQoS values. These QoS values affect the behavior of the DomainParticipant. This routine may fail if the provided **qos** argument is not internally consistent. In this case, ReturnCode_t.RETCODE_INCONSISTENT_POLICY will be returned, and no changes will be made to the DomainParticipant QoS.

## 2.9 Entity Class Reference

### 2.9.1 Description

Base class for all DDS Entities. Inheritance diagram for Entity:




### Public Member Functions

- virtual StatusCondition get_statuscondition ()
- virtual uint get_status_changes ()
- virtual ReturnCode_t enable ()
- virtual InstanceHandle_t get_instance_handle ()

### 2.9.2 Member Function Documentation

#### 2.9.2.1 virtual ReturnCode_t enable ( ) [inline], [virtual]

Enable this Entity. An Entity will begin to communicate only after it is enabled.

Reimplemented in DomainParticipant, Topic, DataReader, Subscriber, DataWriter, and Publisher.

#### 2.9.2.2 virtual InstanceHandle_t get_instance_handle ( ) [inline], [virtual]

Gets the handle that locally identifies this Entity.

Reimplemented in DomainParticipant, Topic, DataReader, Subscriber, DataWriter, and Publisher.

#### 2.9.2.3 virtual uint get_status_changes ( ) [inline], [virtual]

Gets the current status changes from this Entity.

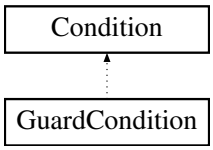
#### 2.9.2.4 virtual StatusCondition get_statuscondition ( ) [inline], [virtual]

Gets the StatusCondition associated with this Entity.

## 2.10 GuardCondition Class Reference

### 2.10.1 Description

A GuardCondition is a Condition where the **trigger_value** is under application control. Inheritance diagram for GuardCondition:




### Public Member Functions

- GuardCondition ()

### 2.10.2 Constructor & Destructor Documentation

#### 2.10.2.1 GuardCondition ( ) [inline]

This routine creates a GuardCondition.

## 2.11 LoggingFlagsKind Class Reference

### 2.11.1 Description

Logging flags

### Public Attributes

- const int COREDX_ERROR_LOGGING_QOS = 0x0001
- const int COREDX_DATA_LOGGING_QOS = 0x0002
- const int COREDX_DISCOVERY_LOGGING_QOS = 0x0004
- const int COREDX_FACTORY_LOGGING_QOS = 0x0008
- const int COREDX_LIVELINESS_LOGGING_QOS = 0x0010
- const int COREDX_STATUS_LOGGING_QOS = 0x0020

### 2.11.2 Member Data Documentation

#### 2.11.2.1 const int COREDX_DATA_LOGGING_QOS = 0x0002

logging flag

#### 2.11.2.2 const int COREDX_DISCOVERY_LOGGING_QOS = 0x0004

logging flag

#### 2.11.2.3 const int COREDX_ERROR_LOGGING_QOS = 0x0001

logging flag

#### 2.11.2.4 const int COREDX_FACTORY_LOGGING_QOS = 0x0008

logging flag

#### 2.11.2.5 const int COREDX_LIVELINESS_LOGGING_QOS = 0x0010

logging flag

#### 2.11.2.6 const int COREDX_STATUS_LOGGING_QOS = 0x0020

logging flag

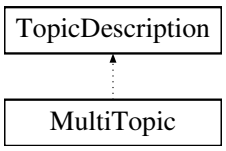
## 2.12 MultiTopic Class Reference

### 2.12.1 Description

DDS.MultiTopic provides a topic that may include data from multiple Topics.

**Not Yet Supported** CoreDX DDS for C# does not yet implement MultiTopics.

Inheritance diagram for MultiTopic:




### Public Member Functions

- DomainParticipant get_participant ()
- String get_type_name ()
- String get_name ()

### 2.12.2 Member Function Documentation

#### 2.12.2.1 String get_name ( ) [inline]

This operation returns **topic_name** of the Topic.

Implements TopicDescription.

#### 2.12.2.2 DomainParticipant get_participant ( ) [inline]

This operation returns the parent DomainParticipant of the Topic.

Implements TopicDescription.

#### 2.12.2.3 String get_type_name ( ) [inline]

This operation returns **type_name** of the Topic.

Implements TopicDescription.

## 2.13 ParticipantBuiltinTopicDataDataReader Class Reference

### 2.13.1 Description

The ParticipantBuiltinTopicDataDataReader is a built-in DataReader that can be used to access Participant Discovery information. The reader can be accessed through the special 'builtin' Subscriber via the DomainParticipant.get_builtin_subscriber() routine.

**See also**

ParticipantBuiltinTopicData
DomainParticipant::get_builtin_subscriber

Inherits DataReaderI< T >.

## 2.14 PublicationBuiltinTopicDataDataReader Class Reference

### 2.14.1 Description

The PublicationBuiltinTopicDataDataReader is a built-in DataReader that can be used to access Publication Discovery information. The reader can be accessed through the special 'builtin' Subscriber via the DomainParticipant.get_builtin_subscriber() routine.

**See also**

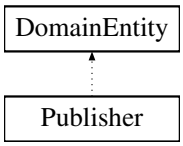
PublicationBuiltinTopicData
DomainParticipant::get_builtin_subscriber

Inherits DataReaderI< T >.

## 2.15 Publisher Class Reference

### 2.15.1 Description

The Publisher configures, creates, manages and destroys DataWriters. Inheritance diagram for Publisher:




### Public Member Functions

- override ReturnCode_t enable ()
- override InstanceHandle_t get_instance_handle ()
- DomainParticipant get_participant ()
- DataWriter create_datawriter (Topic topic, DataWriterQos dw_qos, DataWriterListener listener, uint mask)
- ReturnCode_t delete_datawriter (DataWriter datawriter)
- ReturnCode_t delete_contained_entities ()
- DataWriter lookup_datawriter (String topic_name)
- ReturnCode_t set_qos (PublisherQos qos)
- ReturnCode_t get_qos (PublisherQos qos)
- ReturnCode_t set_listener (PublisherListener new_listener, uint mask)
- PublisherListener get_listener ()
- ReturnCode_t suspend_publications ()
- ReturnCode_t resume_publications ()
- ReturnCode_t begin_coherent_changes ()
- ReturnCode_t end_coherent_changes ()
- ReturnCode_t wait_for_acknowledgments (Duration_t max_wait)
- ReturnCode_t set_default_datawriter_qos (DataWriterQos qos)
- ReturnCode_t get_default_datawriter_qos (DataWriterQos qos)
- ReturnCode_t copy_from_topic_qos (DataWriterQos qos, TopicQos topic_qos)

### 2.15.2 Member Function Documentation

#### 2.15.2.1 ReturnCode_t begin_coherent_changes ( ) [inline]

**Not Yet Supported** This operation is not yet implemented.

#### 2.15.2.2 ReturnCode_t copy_from_topic_qos ( DataWriterQos *qos,* TopicQos *topic_qos* ) [inline]

This operation copies the QoS settings in **a_topic_qos** to the corresponding settings in **a_datawriter_qos.** The **a_datawriter_qos** parameter is populated with a copy of the QoS policies from the **a_topic_qos** structure. QoS entries in the datawriter qos structure will be overwritten with the values from the topic.

#### 2.15.2.3 DataWriter create_datawriter ( Topic *topic,* DataWriterQos *dw_qos,* DataWriterListener *listener,* uint *mask* ) [inline]

This operation creates a DataWriter. The created DataWriter is contained within the Publisher **p.** It is associated with the Topic, ContentFilteredTopic, or MultiTopic indicated by **a_topic,** and has the DataWriterQos indicated by **qos.** The **qos** argument may be passed DDS.DATAWRITER_QOS_DEFAULT, which indicates that the Publisher should use its currently configured default data writer QoS values. The DataWriterListener **a_listener,** is installed at creation time.

The created DataWriter (if not NULL) must be destroyed by a call to Publisher.delete_datawriter().

This routine will fail if the provided QoS settings are internally inconsistent. In this case, the routine will return **NULL.**

#### 2.15.2.4 ReturnCode_t delete_contained_entities ( ) [inline]

This operation deletes all the DataWriters created by means of the Publisher.create_datawriter() operation on the Publisher **p.** This routine will recursively call the corresponding delete_contained_entities() operation on each of the contained DataWriter objects. After successful execution, the application may delete the Publisher by calling DomainParticipant.delete_publisher().

If any of the objects cannot be deleted, this routine will return ReturnCode_t.RETCODE_PRECONDITION_NOT_MET.

#### 2.15.2.5 ReturnCode_t delete_datawriter ( DataWriter *datawriter* ) [inline]

This operation deletes a DataWriter. If the provided DataWriter **a_datawriter** was not created by Publisher **p,** the routine will fail and will return ReturnCode_t.RETCODE_PRECONDITION_NOT_MET.

#### 2.15.2.6 override ReturnCode_t enable ( ) [inline], [virtual]

Enables the Publisher. A Publisher is created either enabled or not based on the DomainParticipantQos setting **entity_factory.** When a Publisher is not enabled, only the following sub-set of all Publisher operations are legal:

- operations to get and set QoS policies,
- factory operations (create, delete),
- get_statuscondition(),
- get_status_changes(),
- lookup operations

Any other operation may return the ReturnCode_t.RETCODE_NOT_ENABLED error. Publisher_enable() may be called on an already enabled Subscriber [it will have no effect].

Reimplemented from Entity.

#### 2.15.2.7 ReturnCode_t end_coherent_changes ( ) [inline]

**Not Yet Supported** This operation is not yet implemented.

#### 2.15.2.8 ReturnCode_t get_default_datawriter_qos ( DataWriterQos *qos* ) [inline]

Provides access to the default DataWriterQos settings held in the Publisher **p.** The provided **qos** argument is populated with the current default qos settings.

#### 2.15.2.9 override InstanceHandle_t get_instance_handle ( ) [inline], [virtual]

Gets the handle that locally identifies this Entity.

Reimplemented from Entity.

#### 2.15.2.10 PublisherListener get_listener ( ) [inline]

This operation returns the currently installed PublisherListener.

#### 2.15.2.11 DomainParticipant get_participant ( ) [inline]

This operation returns the DomainParticipant this Publisher belongs to.

#### 2.15.2.12 ReturnCode_t get_qos ( PublisherQos *qos* ) [inline]

Returns the current PublisherQos settings held in the Publisher **p.** The **qos** parameter is populated with a copy of the current Publisher QoS properties.

#### 2.15.2.13 DataWriter lookup_datawriter ( String *topic_name* ) [inline]

This operation retrieves a previously-created DataWriter contained in the Publisher, attached to a Topic named **topic_name.** If multiple DataWriters are found, one of them will be returned. If no matching DataWriter is found, this routine will return **NULL.**

#### 2.15.2.14 ReturnCode_t resume_publications ( ) [inline]

**Not Yet Supported** This operation is not yet implemented.

#### 2.15.2.15 ReturnCode_t set_default_datawriter_qos ( DataWriterQos *qos* ) [inline]

Sets the default DataWriterQos held in the Publisher. This default qos will be used during subsequent calls to Publisher.create_datawriter() if the special DDS.DATAWRITER_QOS_DEFAULT value is provided for **qos**. This routine may fail if the provided **qos** argument is not internally consistent. In this case, ReturnCode_t.RETCODE_INCONSISTENT_POLICY will be returned, and no changes will be made to the Publisher.

#### 2.15.2.16 ReturnCode_t set_listener ( PublisherListener *new_listener,* uint *mask* ) [inline]

Installs a PublisherListener on Publisher **p.** Only one listener may be attached to a Publisher at a time. A call to **set_listener()** will replace any current listener with **a_listener.**

**a_listener** can be NULL, which indicates a listener that does nothing.

#### 2.15.2.17 ReturnCode_t set_qos ( PublisherQos *qos* ) [inline]

Sets the PublisherQos values. These QoS values affect the behavior of the Publisher. This routine may fail if the provided **qos** argument is not internally consistent. In this case, ReturnCode_t.RETCODE_INCONSISTENT_POLICY will be returned, and no changes will be made to the Publisher QoS.

#### 2.15.2.18 ReturnCode_t suspend_publications ( ) [inline]

**Not Yet Supported** This operation is not yet implemented.

#### 2.15.2.19 ReturnCode_t wait_for_acknowledgments ( Duration_t *max_wait* ) [inline]

Block until all writers contained by this publisher have received acknowledgements.

This routine will block until all data written by contained writers has been acknowledged, or until the 'max_wait' duration has passed. 'max_wait' can be set to INFINITE, in which case this routine may block indefinitely.

**Return values**

| | |
|---|---|
| *DDS_RETCODE_TIME_OUT* | returned if 'max_wait' passes before all acks are received |
| *DDS_RETCODE_OK* | returned if all acks have been received before 'max_wait' |

## 2.16 QosProvider Class Reference

### 2.16.1 Description

QosProvider loads QoS settings from a library and provides interfaces to access entity specific QoS Policies.

### Public Member Functions

- QosProvider (string uri, string profile)

  *Constructor. Creates a new QosProvider instance that loads QoS settings from a library.*
- void cleanup ()

  *"Destructor". Returns a QosProvider back to the factory.*
- DomainParticipantFactoryQos get_participantfactory_qos (string id)

  *Access a DomainParticipantFactory QoS.*
- DomainParticipantQos get_participant_qos (string id)

  *Access a DomainParticipant QoS.*
- TopicQos get_topic_qos (string id)

  *Access a Topic QoS.*
- SubscriberQos get_subscriber_qos (string id)

  *Access a Subscriber QoS.*
- PublisherQos get_publisher_qos (string id)

  *Access a Publisher QoS.*
- DataReaderQos get_datareader_qos (string id)

  *Access a DataReader QoS.*
- DataWriterQos get_datawriter_qos (string id)

  *Access a DataWriter QoS.*

## 2.17 QueryCondition Class Reference

### 2.17.1 Description

The **trigger_value** is driven by the data available, after applying the filter, in the associated DataReader.

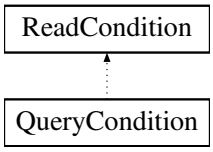
CoreDX DDS fully supports QueryConditions as an argument to DataReader::read_w_condition() and DataReader::take_w_conditions()

**See also**

DataReader::create_querycondition()
FooDataReader::read_w_condition()
FooDataReader::take_w_condition()

**Not Yet Supported** CoreDX DDS does not yet support QueryConditions as triggers for a WaitSet.

Inheritance diagram for QueryCondition:




### Public Member Functions

- String get_query_expression ()
- ReturnCode_t get_query_parameters (List< String > eparams)
- ReturnCode_t set_query_parameters (List< String > parameters)

### 2.17.2 Member Function Documentation

#### 2.17.2.1 String get_query_expression ( ) [inline]

Provides access to the query expression.

The query expression is an SQL syntax conditional expression that is provided when the QueryCondition is created.

**See also**

DataReader::create_querycondition()

#### 2.17.2.2 ReturnCode_t get_query_parameters ( List< String > *eparams* ) [inline]

Provides access to the parameters of the query expression.

The query expression is an SQL syntax conditional expression that is provided when the QueryCondition is created. The query expression may contain references to positional parameters of the form '%0', '%1'. These parameters can be changed dynamically to affect the expression.

**See also**

DataReader::create_querycondition()
QueryCondition::set_query_parameters()

#### 2.17.2.3 ReturnCode_t set_query_parameters ( List< String > *parameters* ) [inline]

Modifies the parameters of the query expression.

The **query_expression** is an SQL like condition expression, and the **parameters** argument provides optional parameters that are referenced by the **query_expression**. The syntax for refering to paramters in a query_expression is the percent sign " followed by a number. The number is the index of the paramter in the **query_paramters** sequence. Parameters are counted starting at zero. So, "%0" refers to the first parameter, and "%4" refers to the fifth paramter. Using this syntax, the expression "x<%0" would test the value of 'x' against the first parameter in the sequence.

This routine allows the parameters to be changed dynamically.

**See also**

DataReader::create_querycondition()

## 2.18 ReadCondition Class Reference

### 2.18.1 Description

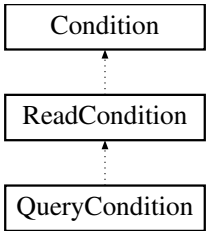
A ReadCondition is a specialized Condition associated with a DataReader.

The **trigger_value** is driven by the data available in the associated DataReader. A ReadCondition is obtained by calling the DataReader_create_readcondition() function. When the ReadCondition is no longer needed, it should be destroyed by a call to DataReader.delete_readcondition().

**See also**

DataReader::create_readcondition(long sample_states, long view_states, long instance_states)
DataReader::delete_readcondition(ReadCondition)

Inheritance diagram for ReadCondition:




### Public Member Functions

- uint get_sample_state_mask ()
- uint get_view_state_mask ()
- uint get_instance_state_mask ()
- DataReader get_datareader ()

### 2.18.2 Member Function Documentation

#### 2.18.2.1 DataReader get_datareader ( ) [inline]

Gets the single DataReader associated with this ReadCondition.

#### 2.18.2.2 uint get_instance_state_mask ( ) [inline]

Gets the current value of the instance_state_mask in this ReadCondition.

#### 2.18.2.3 uint get_sample_state_mask ( ) [inline]

Gets the current value of the sample_state_mask in this ReadCondition.

#### 2.18.2.4 uint get_view_state_mask ( ) [inline]

Gets the current value of the view_state_mask in this ReadCondition.

## 2.19 SampleInfo Class Reference

### 2.19.1 Description

The SampleInfo structure contains information associated with each Sample. The DataReader.read() and take() operations return two vectors. One vector contains Sample(s) and the other contains SampleInfo(s). There is a one-to-one correspondence between items in these two vectors. Each Sample is described by the corresponding SampleInfo instance.

### Public Attributes

- uint sample_state
- uint view_state
- uint instance_state
- Time_t source_timestamp
- Time_t reception_timestamp
- InstanceHandle_t instance_handle
- InstanceHandle_t publication_handle
- int disposed_generation_count
- int no_writers_generation_count
- int sample_rank
- int generation_rank
- int absolute_generation_rank
- bool valid_data

### 2.19.2 Member Data Documentation

#### 2.19.2.1 int absolute_generation_rank

The generation difference between this sample and the most recent sample. The **absolute_generation_rank** indicates the generation difference (ie, the number of times the instance was disposed and became alive again) between this sample, and the most recent sample (possibly not in the returned collection) of this instance.

#### 2.19.2.2 int disposed_generation_count

The number of times the instance has become 'ALIVE' after being explicitly disposed. (Computed at the time the sample arrives at the DataReader.)

#### 2.19.2.3 int generation_rank

The generation difference of this sample and the most recent sample in the collection. **generation_rank** indicates the generation difference (ie, the number of times the instance was disposed and became alive again) between this sample, and the most recent sample in the collection related to this instance.

#### 2.19.2.4 InstanceHandle_t instance_handle

The handle that locally identifies the associated instance.

#### 2.19.2.5 uint instance_state

Indicates whether the associated instance currently exists. **instance_state** can be one of:
DDS.ALIVE_INSTANCE_STATE
DDS.NOT_ALIVE_DISPOSED_INSTANCE_STATE
DDS.NOT_ALIVE_NO_WRITERS_INSTANCE_STATE

Can use DDS.NOT_ALIVE_INSTANCE_STATE as a test for either 'NOT_ALIVE' state. For example if (sample_info->view_state & DDS.NOT_ALIVE_INSTANCE_STATE) ...

#### 2.19.2.6 int no_writers_generation_count

The number of times the instance has become 'ALIVE' after being automatically disposed due to no active writers. (Computed at the time the sample arrives at the DataReader.)

#### 2.19.2.7 InstanceHandle_t publication_handle

The local handle of the source DataWriter.

#### 2.19.2.8 Time_t reception_timestamp

The time when the sample was received by the DataReader.

#### 2.19.2.9 int sample_rank

Number of samples related to this instances that follow in the collection returned by the DataReader **read** or **take** operations.

#### 2.19.2.10 uint sample_state

The associated data sample has/has not been read previously.

**sample_state** indicates whether or not the DataReader has previously read the associated sample.
One of:
DDS.READ_SAMPLE_STATE
DDS.NOT_READ_SAMPLE_STATE.

Use DDS.ANY_SAMPLE_STATE to test for either state.

#### 2.19.2.11 Time_t source_timestamp

The time provided by the DataWriter when the sample was written.

#### 2.19.2.12 bool valid_data

Is set to true if the associated DataSample contains data. The associated DataSample may not contain data if it this sample indicates a change in sample state (for example ALIVE -> DISPOSED).

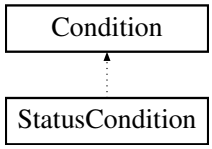
#### 2.19.2.13 uint view_state

Associated instance has/has not been seen before. **view_state** indicates whether the DataReader has already seen samples for the most current generation of the related instance.
**view_state** can be one of:
DDS.NEW_VIEW_STATE
DDS.NOT_NEW_VIEW_STATE

## 2.20 StatusCondition Class Reference

### 2.20.1 Description

A StatusCondition is a condition associated with an Entity. The **trigger_value** is driven by the communication status of the associated Entity. Inheritance diagram for StatusCondition:




### Public Member Functions

- ReturnCode_t set_enabled_statuses (uint mask)
- uint get_enabled_statuses ()
- Entity get_entity ()

### 2.20.2 Member Function Documentation

#### 2.20.2.1 uint get_enabled_statuses ( ) [inline]

This routine returns the statuses which are enabled in this StatusCondition. The statuses are returned as a bitmask.

**See also**

DDS::ALL_STATUS

#### 2.20.2.2 Entity get_entity ( ) [inline]

This routine returns the single entity associated with this StatusCondition.

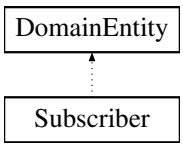
#### 2.20.2.3 ReturnCode_t set_enabled_statuses ( uint *mask* ) [inline]

This routine sets the statuses which are enabled in this StatusCondition. The statuses are provided as a bitmask.

## 2.21 Subscriber Class Reference

### 2.21.1 Description

The Subscriber configures, creates, manages and destroys DataReaders. Inheritance diagram for Subscriber:




#### Public Member Functions

- override ReturnCode_t enable ()
- override InstanceHandle_t get_instance_handle ()
- DataReader create_datareader (TopicDescription topic, DataReaderQos dr_qos, DataReaderListener listener, uint mask)
- ReturnCode_t delete_datareader (DataReader datareader)
- ReturnCode_t delete_contained_entities ()
- DataReader lookup_datareader (String topic_name)
- ReturnCode_t get_datareaders (List< DataReader > readers, long sample_states, long view_states, long instance_states)
- DomainParticipant get_participant ()
- ReturnCode_t set_qos (SubscriberQos qos)
- ReturnCode_t get_qos (SubscriberQos qos)
- ReturnCode_t set_listener (SubscriberListener new_listener, uint mask)
- SubscriberListener get_listener ()
- ReturnCode_t begin_access ()
- ReturnCode_t end_access ()
- ReturnCode_t set_default_datareader_qos (DataReaderQos qos)
- ReturnCode_t get_default_datareader_qos (DataReaderQos qos)
- ReturnCode_t copy_from_topic_qos (DataReaderQos qos, TopicQos topic_qos)

### 2.21.2 Member Function Documentation

#### 2.21.2.1 ReturnCode_t begin_access ( ) [inline]

This operation indicates that the application is about to access the data samples in any of the DataReader objects contained in the Subscriber **s.** The application is expected to use this operation only if PRESENTATION QosPolicy of the Subscriber has the access_scope set to GROUP. [Otherwise this routine has no effect.]

**Not Yet Supported** This is currently unsupported in the C# language binding.

#### 2.21.2.2 ReturnCode_t copy_from_topic_qos ( DataReaderQos *qos,* TopicQos *topic_qos* ) [inline]

This operation copies the QoS settings in **a_topic_qos** to the corresponding settings in **a_datareader_qos.** The **a_datareader_qos** parameter is populated with a copy of the QoS policies from the **a_topic_qos** structure. QoS entries in the datareader qos structure will be overwritten with the values from the topic.

#### 2.21.2.3 DataReader create_datareader ( TopicDescription *topic,* DataReaderQos *dr_qos,* DataReaderListener *listener,* uint *mask* ) [inline]

This operation creates a DataReader. The created DataReader is contained within the Subscriber **s.** It is associated with the Topic, ContentFilteredTopic, or MultiTopic indicated by **a_topic,** and has the DataReaderQos indicated by **qos.** The **qos** argument may be passed DDS.DATAREADER_QOS_DEFAULT, which indicates that the Subscriber should use its currently configured default data reader QoS values. The DataReaderListener **a_listener,** is installed at creation time.

The created DataReader (if not NULL) must be destroyed by a call to Subscriber.delete_datareader().

This routine will fail if the provided QoS settings are internally inconsistent. In this case, the routine will return **NULL.**

#### 2.21.2.4 ReturnCode_t delete_contained_entities ( ) [inline]

This operation deletes all the DataReaders created by means of the Subscriber.create_datareader() operation on the Subscriber **s.** This routine will recursively call the corresponding delete_contained_entities() operation on each of the contained DataReader objects. If successful, this operation will recursively delete all objects contained with this Subscriber. After successful execution, the application may delete the Subscriber by calling DomainParticipant.delete_subscriber().

If any of the objects cannot be deleted, this routine will return ReturnCode_t.RETCODE_PRECONDITION_NOT_MET.

#### 2.21.2.5 ReturnCode_t delete_datareader ( DataReader *datareader* ) [inline]

This operation deletes a DataReader. If the provided DataReader **a_datareader** was not created by Subscriber **s,** the routine will fail and will return ReturnCode_t.RETCODE_PRECONDITION_NOT_MET. If the indicated DataReader has any outstanding ReadCondition or QueryCondition objects the routine will fail and will return ReturnCode_t.RETCODE_PRECONDITION_NOT_MET. If the indicated DataReader has any outstanding 'loans' (from read() or take() operations), the routine will fail and will return ReturnCode_.tRETCODE_PRECONDITION_NOT_MET.

#### 2.21.2.6 override ReturnCode_t enable ( ) [inline], [virtual]

Enables the Subscriber. A Subscriber is created either enabled or not based on the DomainParticipantQos setting **entity_factory.** When a Subscriber is not enabled, only the following sub-set of all Subscriber operations are legal:

- operations to get and set QoS policies,
- factory operations (create, delete),
- get_statuscondition(),
- get_status_changes(),
- lookup operations

Any other Subscriber operation may return the ReturnCode_t.RETCODE_NOT_ENABLED error. Subscriber_enable() may be called on an already enabled Subscriber [it will have no effect].

Reimplemented from Entity.

#### 2.21.2.7 ReturnCode_t end_access ( ) [inline]

This operation closes a corresponding Subscriber.begin_access().

**Not Yet Supported** This is currently unsupported in the C# language binding.

#### 2.21.2.8 ReturnCode_t get_datareaders ( List< DataReader > *readers,* long *sample_states,* long *view_states,* long *instance_states* ) [inline]

This operation allows the application to access DataReader objects that contain samples with the specified **sample_states, view_states,** and **instance_states.**

If the PRESENTATION QosPolicy of the Subscriber to which the DataReader belongs has the access_scope set to GROUP, this operation should be invoked only inside a begin_access/end_access block. Otherwise it will return the error PRECONDITION_NOT_MET.

The returned collection of DataReader objects may either be a set (containing each DataReader at most once in no specified order), or a list (containing each DataReader one or more times in a specific order).

1. If PRESENTATION access_scope is INSTANCE or TOPIC, the returned collection is a set.
2. If PRESENTATION access_scope is GROUP and ordered_access is set to TRUE, then the returned collection is a list.

This difference supports alternate access mechanisms.

**Not Yet Supported** This is currently unsupported in the C# language binding.

#### 2.21.2.9 ReturnCode_t get_default_datareader_qos ( DataReaderQos *qos* ) [inline]

Provides access to the default DataReaderQos settings held in the Subscriber **s.** The provided **qos** argument is populated with the current default qos settings.

#### 2.21.2.10 override InstanceHandle_t get_instance_handle ( ) [inline], [virtual]

Gets the handle that locally identifies this Entity.

Reimplemented from Entity.

#### 2.21.2.11 SubscriberListener get_listener ( ) [inline]

This operation returns the currently installed SubscriberListener.

#### 2.21.2.12 DomainParticipant get_participant ( ) [inline]

This operation returns the DomainParticipant this Subscriber belongs to.

#### 2.21.2.13 ReturnCode_t get_qos ( SubscriberQos *qos* ) [inline]

Returns the current SubscriberQos settings held in the Subscriber **s.** The **qos** parameter is populated with a copy of the current Subscriber QoS properties.

#### 2.21.2.14 DataReader lookup_datareader ( String *topic_name* ) [inline]

This operation retrieves a previously-created DataReader contained in the Subscriber, attached to a Topic named **topic_name.** If multiple DataReaders are found, one of them will be returned. If no matching DataReader is found, this routine will return **NULL.**

This routine is useful to obtain access to a particular built-in DataReader.

#### 2.21.2.15 ReturnCode_t set_default_datareader_qos ( DataReaderQos *qos* ) [inline]

Sets the default DataReaderQos held in the Subscriber. This default qos will be used during subsequent calls to Subscriber.create_datareader() if the special DDS.DATAREADER_QOS_DEFAULT value is provided for **qos**. This routine may fail if the provided **qos** argument is not internally consistent. In this case, ReturnCode_t.RETCODE_INCONSISTENT_POLICY will be returned, and no changes will be made to the Subscriber.

#### 2.21.2.16 ReturnCode_t set_listener ( SubscriberListener *new_listener,* uint *mask* ) [inline]

Installs a SubscriberListener on Subscriber **s.** Only one listener may be attached to a Subscriber at a time. A call to **set_listener()** will replace any current listener with **a_listener.**

**a_listener** can be NULL, which indicates a listener that does nothing.

#### 2.21.2.17 ReturnCode_t set_qos ( SubscriberQos *qos* ) [inline]

Sets the SubscriberQos values. These QoS values affect the behavior of the Subscriber. This routine may fail if the provided **qos** argument is not internally consistent. In this case, ReturnCode_t.RETCODE_INCONSISTENT_POLICY will be returned, and no changes will be made to the Subscriber QoS.

## 2.22 SubscriptionBuiltinTopicDataDataReader Class Reference

### 2.22.1 Description

The SubscriptionBuiltinTopicDataDataReader is a built-in DataReader that can be used to access Subscription Discovery information. The reader can be accessed through the special 'builtin' Subscriber via the DomainParticipant.get_builtin_subscriber() routine.

**See also**

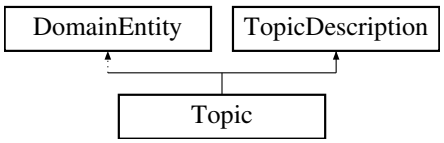
SubscriptionBuiltinTopicData
DomainParticipant::get_builtin_subscriber

Inherits DataReaderI< T >.

## 2.23 Topic Class Reference

### 2.23.1 Description

Topic is the basic description of data to be published or subscribed. A topic is identified by a **name** and a **type.** A Topic is created by calling DomainParticipant.create_topic(). Prior to creating a Topic, the associated data type must be registered with the DomainParticipant via a call to the TypeSupportXYZ.register_type() function. [The register_type() function is auto-generated 'type-specific' code.] Inheritance diagram for Topic:




### Public Member Functions

- DomainParticipant get_participant ()
- String get_type_name ()
- String get_name ()
- override ReturnCode_t enable ()
- override InstanceHandle_t get_instance_handle ()
- ReturnCode_t set_qos (TopicQos qos)
- ReturnCode_t get_qos (TopicQos qos)
- ReturnCode_t set_listener (TopicListener new_listener, uint mask)
- TopicListener get_listener ()
- ReturnCode_t get_inconsistent_topic_status (out InconsistentTopicStatus status)

### 2.23.2 Member Function Documentation

#### 2.23.2.1 override ReturnCode_t enable ( ) [inline], [virtual]

Enables the Topic. A Topic is created either enabled or not based on the DomainParticipantQos setting **entity_factory.** When a Topic is not enabled, only the following sub-set of all Topic operations are legal:

- operations to get and set QoS policies,
- get_name(), get_type_name(), get_participant()
- get_statuscondition(),
- get_status_changes(),

Any other operation may return the ReturnCode_t.RETURNCODE_NOT_ENABLED error. Topic.enable() may be called on an already enabled Topic [it will have no effect].

Reimplemented from Entity.

#### 2.23.2.2 ReturnCode_t get_inconsistent_topic_status ( out InconsistentTopicStatus *status* ) [inline]

Provides access to the InconsistentTopicStatus of the Topic. As a side-effect, this routine will reset the **total_count_change** status field to zero.

#### 2.23.2.3 override InstanceHandle_t get_instance_handle ( ) [inline], [virtual]

Gets the handle that locally identifies this Entity.

Reimplemented from Entity.

#### 2.23.2.4 TopicListener get_listener ( ) [inline]

This operation returns the currently installed TopicListener.

#### 2.23.2.5 String get_name ( ) [inline]

This operation returns **topic_name** of the Topic.

Implements TopicDescription.

#### 2.23.2.6 DomainParticipant get_participant ( ) [inline]

This operation returns the parent **DomainParticipant** of the Topic.

Implements TopicDescription.

#### 2.23.2.7 ReturnCode_t get_qos ( TopicQos *qos* ) [inline]

Returns the current TopicQos settings held in the Topic **t.** This routine copies the Topic QoS properites into **qos.**

#### 2.23.2.8 String get_type_name ( ) [inline]

This operation returns **type_name** of the Topic.

Implements TopicDescription.

#### 2.23.2.9 ReturnCode_t set_listener ( TopicListener *new_listener,* uint *mask* ) [inline]

Installs a TopicListener on Topic **t.** Only one listener may be attached to a Topic at a time. A call to **set_listener()** will replace any current listener with **a_listener.**

**a_listener** can be NULL, which indicates a listener that does nothing.

The infrastructure will make an internal copy of the listener structure so that it need not be persisted by the application.

#### 2.23.2.10 ReturnCode_t set_qos ( TopicQos *qos* ) [inline]

Sets the TopicQos values. These QoS values affect the behavior of the Topic. This routine may fail if the provided **qos** argument is not internally consistent. In this case, ReturnCode_t.RETCODE_INCONSISTENT_POLICY will be returned, and no changes will be made to the Topic QoS.

## 2.24 WaitSet Class Reference

### 2.24.1 Description

A DDS_WaitSet maintains a set of Condition objects and allows the application to wait until one or more of them have a **trigger_value** of TRUE.

Multiple conditions may be attached to a WaitSet.

**See also**

Condition

### Public Member Functions

- WaitSet ()
- void destroy ()
- ReturnCode_t attach_condition (Condition c)
- ReturnCode_t detach_condition (Condition c)
- ReturnCode_t wait (List< Condition > active_conditions, Duration_t timeout)
- ReturnCode_t get_conditions (List< Condition > attached_conditions)

### 2.24.2 Constructor & Destructor Documentation

#### 2.24.2.1 WaitSet ( ) [inline]

Constructor.

### 2.24.3 Member Function Documentation

#### 2.24.3.1 ReturnCode_t attach_condition ( Condition *c* ) [inline]

Adds the condition **c** to the WaitSet. If another thread is currently 'waiting' on the WaitSet, this newly added condition will unblock that thread if its **trigger_value** is TRUE.

#### 2.24.3.2 void destroy ( ) [inline]

Destructor. Releases internal resources associated with the WaitSet. This WaitSet is destroyed, and must not be used after this call returns.

#### 2.24.3.3 ReturnCode_t detach_condition ( Condition *c* ) [inline]

Adds the condition **c** to the WaitSet. If another thread is currently 'waiting' on the WaitSet, this newly added condition will unblock that thread if its **trigger_value** is TRUE.

#### 2.24.3.4 ReturnCode_t get_conditions ( List< Condition > *attached_conditions* ) [inline]

Retrieves the current list of attached conditions. Populates the **attached_conditions** sequence.

#### 2.24.3.5 ReturnCode_t wait ( List< Condition > *active_conditions,* Duration_t *timeout* ) [inline]

Causes the controlling thread to block until an attached condition is triggered or **timeout** elapses.

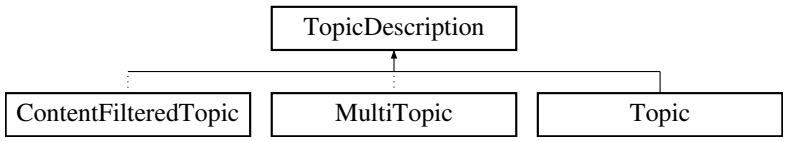
A return value of DDS_RETCODE_OK indicates that one or more of the attached conditions evaluated to TRUE. Those 'active' conditions are included in the 'out' parameter **active_conditions.**

A return value of DDS_RETCODE_TIMEOUT indicates that the timeout period elapsed without any of the conditions evaluating to TRUE.

## 2.25 TopicDescription Interface Reference

### 2.25.1 Description

TopicDescription is an interface that provides the foundation for Topic, ContentFilteredTopic, and MultiTopic. Inheritance diagram for TopicDescription:




### Public Member Functions

- DomainParticipant get_participant ()
- String get_type_name ()
- String get_name ()

### 2.25.2 Member Function Documentation

#### 2.25.2.1 String get_name ( )

This operation returns **topic_name** of the Topic.

Implemented in MultiTopic, ContentFilteredTopic, and Topic.

#### 2.25.2.2 DomainParticipant get_participant ( )

This operation returns the parent **DomainParticipant** of the Topic.

Implemented in MultiTopic, ContentFilteredTopic, and Topic.

#### 2.25.2.3 String get_type_name ( )

This operation returns **type_name** of the Topic.

Implemented in MultiTopic, ContentFilteredTopic, and Topic.

## 2.26 T Class Reference

### 2.26.1 Description

Sample holds the data and related meta information.

Sample will allocate space for the sample data, and maintain that memory. The sample also includes SampleInfo which will be populated if the Sample instance is provided by a read() or similar operation of a DDS or RPC entity.

Inherits new, DdsType, and DdsType.

### Public Member Functions

- ReturnCode_t get_key_value (T key_holder, InstanceHandle_t handle)
- InstanceHandle_t lookup_instance (T instance_data)
- ReturnCode_t return_loan (List< T > received_data, List< SampleInfo > sample_infos)
- ReturnCode_t read (List< T > received_data, List< SampleInfo > sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states)
- ReturnCode_t take (List< T > received_data, List< SampleInfo > sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states)
- ReturnCode_t read_w_condition (List< T > received_data, List< SampleInfo > sample_infos, int max_samples, ReadCondition condition)
- ReturnCode_t take_w_condition (List< T > received_data, List< SampleInfo > sample_infos, int max_samples, ReadCondition condition)
- ReturnCode_t read_next_sample (T received_data, SampleInfo sample_info)
- ReturnCode_t take_next_sample (T received_data, SampleInfo sample_info)
- ReturnCode_t read_instance (List< T > received_data, List< SampleInfo > sample_infos, int max_samples, InstanceHandle_t handle, uint sample_states, uint view_states, uint instance_states)
- ReturnCode_t take_instance (List< T > received_data, List< SampleInfo > sample_infos, int max_samples, InstanceHandle_t handle, uint sample_states, uint view_states, uint instance_states)
- ReturnCode_t read_next_instance (List< T > received_data, List< SampleInfo > sample_infos, int max_samples, InstanceHandle_t prev_handle, uint sample_states, uint view_states, uint instance_states)
- ReturnCode_t take_next_instance (List< T > received_data, List< SampleInfo > sample_infos, int max_samples, InstanceHandle_t prev_handle, uint sample_states, uint view_states, uint instance_states)
- ReturnCode_t read_next_instance_w_condition (List< T > received_data, List< SampleInfo > sample_infos, int max_samples, InstanceHandle_t handle, ReadCondition condition)
- ReturnCode_t take_next_instance_w_condition (List< T > received_data, List< SampleInfo > sample_infos, int max_samples, InstanceHandle_t handle, ReadCondition condition)
- ReturnCode_t get_key_value (T key_holder, InstanceHandle_t handle)
- InstanceHandle_t lookup_instance (T instance_data)
- InstanceHandle_t register_instance (T instance_data)
- InstanceHandle_t register_instance_w_timestamp (T instance_data, Time_t ts)
- ReturnCode_t unregister_instance (T instance_data, InstanceHandle_t handle)
- ReturnCode_t unregister_instance_w_timestamp (T instance_data, InstanceHandle_t handle, Time_t timestamp)
- ReturnCode_t write (T instance_data, InstanceHandle_t handle)
- ReturnCode_t write_w_timestamp (T instance_data, InstanceHandle_t handle, Time_t timestamp)
- ReturnCode_t dispose (T instance_data, InstanceHandle_t handle)
- ReturnCode_t dispose_w_timestamp (T instance_data, InstanceHandle_t handle, Time_t timestamp)

### Static Public Member Functions

- static implicit operator T (Sample< T > s)

### 2.26.2 Member Function Documentation

#### 2.26.2.1 ReturnCode_t dispose ( T *instance_data,* InstanceHandle_t *handle* )

Indicates that the instance no longer exists. If **handle** is not HANDLE_NIL, then **handle** must identify a valid instance that has been previously registered or written by this DataWriter. The current time is used as the source timstamp.

#### 2.26.2.2 ReturnCode_t dispose_w_timestamp ( T *instance_data,* InstanceHandle_t *handle,* Time_t *timestamp* )

Indicates that the instance no longer exists. If **handle** is not HANDLE_NIL, then **handle** must identify a valid instance that has been previously registered or written by this DataWriter. The **source_timestamp** is used as the source timstamp for the published message.

#### 2.26.2.3 ReturnCode_t get_key_value ( T *key_holder,* InstanceHandle_t *handle* )

This routine will populate the data structure indicated by **key_holder** with the key infomation identified by **handle.**

This routine is data type specific. The generated type specific DataWriter includes an implementation of this routine which should be used to support type-safety.

#### 2.26.2.4 ReturnCode_t get_key_value ( T *key_holder,* InstanceHandle_t *handle* )

This routine will populate the data structure indicated by **key_holder** with the key infomation identified by **handle.**

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

#### 2.26.2.5 InstanceHandle_t lookup_instance ( T *instance_data* )

Returns the handle that identifies the data instance provided in **instance_data.** The 'key' field values of the data are associated with a unique handle.

#### 2.26.2.6 InstanceHandle_t lookup_instance ( T *instance_data* )

Returns the handle that identifies the data instance provided in **instance_data.** The 'key' field values of the data are associated with a unique handle.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

#### 2.26.2.7 static implicit operator T ( Sample< T > *s* ) [inline],[static]

implicitly cast from Sample<T> to T

#### 2.26.2.8 ReturnCode_t read ( List< T > *received_data,* List< SampleInfo > *sample_infos,* int *max_samples,* uint *sample_states,* uint *view_states,* uint *instance_states* )

This operation accesses a collection of data values (with associated SampleInfo) within the DataReader. This routine, and the related **take(),** provide the interface for an application to access published data. There are several varieties of **read()** and **take(),** to facilitate different access patterns or approaches.

The primary difference between **read()** and **take()** is that **take()** removes all returned data samples from the DataReader while **read()** does not. Sequential **read()** calls will return the same data samples each time (if nothing else changes); while sequential **take()** calls will return data samples for only the first call. Subsequent **take()** calls will return an empty collection (if no new data arrives).

The specific behavior of **read()** depends on several things: input paramters, the QoS settings of the DataReader, and the state of recieved data. First, the **received_data** and **sample_infos** arguments affect the following:

- how many samples are returned, and
- whether the returned data should be 'loaned' or copied.

The argument **max_samples** is used to further limit the number of samples returned.

The **sample_states, view_states,** and **instance_states** arguments are used to selectively add data samples to the returned collections. These arguments indicate the desired 'states' for data samples and instances. These state arguments are bit masks; they can be the bit-wise OR of several individual state flags or they may use the special 'ANY' constants (e.g.: DDS.ANY_SAMPLE_STATE). Only samples that have a matching state for all three categories are added to the returned collection.

The order of samples in the returned collections is determined by the **PRESENTATION** and **DESTINATION_ORDER** QoS policies.

The returned collection is held in **recieved_data** and **samples_infos.** These two sequences operate together to represent a sequence of pairs (data, SampleInfo). Each data item in **received_data** has a corresponding entry in **sample_infos** that provides associated 'meta-data'. See SampleInfo for a description of this meta-data.

In CoreDX DDS, the returned sequences contain 'loaned' data. This provides zero-copy access to the data, and provides a very efficient data access mechanism. Becuase the data is 'loaned' to the application, the application is required to indicate when it is finished accessing the data. This is accomplished by calling return_loan().

The **read()** operation will set the SampleInfo.sample_state to DDS.READ_SAMPLE_STATE.

The **read()** operation may set the SampleInfo.view_state to DDS.NOT_NEW_VIEW_STATE, if a sample of the most recent generation of the instance is read.

If there is no data found, then the **read()** will return RETCODE_NO_DATA.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

#### 2.26.2.9 ReturnCode_t read_instance ( List< T > *received_data,* List< SampleInfo > *sample_infos,* int *max_samples,* InstanceHandle_t *handle,* uint *sample_states,* uint *view_states,* uint *instance_states* )

This operation accesses a collection of data values (with associated SampleInfo), belonging to a particular instance, within the DataReader.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

**See also**

com.toc.coredx.DDS.TDataReader::read(List<T> received_data, List<SampleInfo> sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states) read()

#### 2.26.2.10 ReturnCode_t read_next_instance ( List< T > *received_data,* List< SampleInfo > *sample_infos,* int *max_samples,* InstanceHandle_t *prev_handle,* uint *sample_states,* uint *view_states,* uint *instance_states* )

This operation accesses a collection of data values (with associated SampleInfo), instance by instance, within the DataReader. To start,pass HANDLE_NIL for parameter prev_handle. After the last instance (in order) has been taken, this routine will return RETCODE_NO_DATA.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

**See also**

com.toc.coredx.DDS.TDataReader::read(List<T> received_data, List<SampleInfo> sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states) read()

#### 2.26.2.11 ReturnCode_t read_next_instance_w_condition ( List< T > *received_data,* List< SampleInfo > *sample_infos,* int *max_samples,* InstanceHandle_t *handle,* ReadCondition *condition* )

This operation accesses a collection of data values (with associated SampleInfo), instance by instance subject to a filter, within the DataReader.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

**See also**

com.toc.coredx.DDS.TDataReader::read(List<T> received_data, List<SampleInfo> sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states) read()

#### 2.26.2.12 ReturnCode_t read_next_sample ( T *received_data,* SampleInfo *sample_info* )

This operation accesses a data value (with associated SampleInfo) within the DataReader.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

**See also**

com.toc.coredx.DDS.TDataReader::read(List<T> received_data, List<SampleInfo> sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states) read()

#### 2.26.2.13 ReturnCode_t read_w_condition ( List< T > *received_data,* List< SampleInfo > *sample_infos,* int *max_samples,* ReadCondition *condition* )

This operation accesses a collection of data values (with associated SampleInfo), subject to a filter, within the DataReader.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

**See also**

com.toc.coredx.DDS.TDataReader::read(List<T> received_data, List<SampleInfo> sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states) read()

#### 2.26.2.14 InstanceHandle_t register_instance ( T *instance_data* )

Declares the existance of an instance identified by the 'key fields' in **instance_data.** The handle that uniquely identifies the instance is returned. The current time is used as the source timestamp.

#### 2.26.2.15 InstanceHandle_t register_instance_w_timestamp ( T *instance_data,* Time_t *ts* )

Declares the existance of an instance identified by the 'key fields' in **instance_data.** The handle that uniquely identifies the instance is returned. The **source_timestamp** is used as the source timestamp.

#### 2.26.2.16 ReturnCode_t return_loan ( List< T > *received_data,* List< SampleInfo > *sample_infos* )

Returns data and sample_info values to a DataReader. When an application calls DataReader **read()** or **take()** operations, these routines 'loan' data and SampleInfo values to the application. [This is an optimization that avoids extra copies of data.] The appliction must return this 'loaned' data to the DataReader. The return_loan() routine indicates to the DataReader that the application no longer requires access to the data and sample_infos.

A call to **return_loan()** operation must be made only if previous **read()** or **take()** calls 'loaned' data to the application. See read() for a discussion of when data is 'loaned'.

If the **received_data** or **sample_infos** parameters provided do not identify data obtained from DataReader **dr,** then the error RETCODE_PRECONDITION_NOT_MET will be returned.

A DataReader cannot be deleted if any 'loans' are outstanding.

**See also**

com.toc.coredx.DDS.TDataReader::read(TSeq received_data, List<SampleInfo> sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states) read()

#### 2.26.2.17 ReturnCode_t take ( List< T > *received_data,* List< SampleInfo > *sample_infos,* int *max_samples,* uint *sample_states,* uint *view_states,* uint *instance_states* )

This operation takes a collection of data values (with associated SampleInfo) from the DataReader. This routine, and the related **read()**, provide the interface for an application to access published data. There are several varieties of **read()** and **take()**, to facilitate different access patterns or approaches.

The primary difference between **read()** and **take()** is that **take()** removes all returned data samples from the DataReader while **read()** does not. Sequential **read()** calls will return the same data samples each time (if nothing else changes); while sequential **take()** calls will return data samples for only the first call. Subsequent **take()** calls will return an empty collection (if no new data arrives).

The specific behavior of **take()** depends on several things: input paramters, the QoS settings of the DataReader, and the state of recieved data. First, the **received_data** and **sample_infos** arguments affect the following:

- how many samples are returned, and
- whether the returned data should be 'loaned' or copied.

The argument **max_samples** is used to further limit the number of samples returned.

The **sample_states, view_states,** and **instance_states** arguments are used to selectively add data samples to the returned collections. These arguments indicate the desired 'states' for data samples and instances. These state arguments are bit masks; they can be the bit-wise OR of several individual state flags or they may use the special 'ANY'

constants (e.g.: DDS.ANY_SAMPLE_STATE). Only samples that have a matching state for all three categories are added to the returned collection.

The order of samples in the returned collections is determined by the **PRESENTATION** and **DESTINATION_ORDER** QoS policies.

The returned collection is held in **recieved_data** and **samples_infos.** These two sequences operate together to represent a sequence of pairs (data, SampleInfo). Each data item in **received_data** has a corresponding entry in **sample_infos** that provides associated 'meta-data'. See SampleInfo for a description of this meta-data.

In CoreDX DDS, the returned sequences contain 'loaned' data. This provides zero-copy access to the data, and provides a very efficient data access mechanism. Becuase the data is 'loaned' to the application, the application is required to indicate when it is finished accessing the data. This is accomplished by calling return_loan().

The **take()** operation will set the SampleInfo.sample_state to DDS.READ_SAMPLE_STATE.

The **take()** operation may set the SampleInfo.view_state to DDS.NOT_NEW_VIEW_STATE, if a sample of the most recent generation of the instance is taken.

If there is no data found, then the **take()** will return RETCODE_NO_DATA.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

#### 2.26.2.18 ReturnCode_t take_instance ( List< T > *received_data,* List< SampleInfo > *sample_infos,* int *max_samples,* InstanceHandle_t *handle,* uint *sample_states,* uint *view_states,* uint *instance_states* )

This operation takes a collection of data values (with associated SampleInfo), belonging to a particular instance, from the DataReader.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

**See also**

com.toc.coredx.DDS.TDataReader::take(List<T> received_data, List<SampleInfo> sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states) take()

#### 2.26.2.19 ReturnCode_t take_next_instance ( List< T > *received_data,* List< SampleInfo > *sample_infos,* int *max_samples,* InstanceHandle_t *prev_handle,* uint *sample_states,* uint *view_states,* uint *instance_states* )

This operation takes a collection of data values (with associated SampleInfo), instance by instance, from the DataReader. To start,pass HANDLE_NIL for parameter prev_handle. After the last instance (in order) has been taken, this routine will return RETCODE_NO_DATA.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

**See also**

com.toc.coredx.DDS.TDataReader::take(List<T> received_data, List<SampleInfo> sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states) take()

#### 2.26.2.20 ReturnCode_t take_next_instance_w_condition ( List< T > *received_data,* List< SampleInfo > *sample_infos,* int *max_samples,* InstanceHandle_t *handle,* ReadCondition *condition* )

This operation takes a collection of data values (with associated SampleInfo), instance by instance subject to a filter, from the DataReader.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

**See also**

com.toc.coredx.DDS.TDataReader::take(List<T> received_data, List<SampleInfo> sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states) take()

#### 2.26.2.21 ReturnCode_t take_next_sample ( T *received_data,* SampleInfo *sample_info* )

This operation takes a data value (with associated SampleInfo) from the DataReader.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

**See also**

com.toc.coredx.DDS.TDataReader::take(List<T> received_data, List<SampleInfo> sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states) take()

#### 2.26.2.22 ReturnCode_t take_w_condition ( List< T > *received_data,* List< SampleInfo > *sample_infos,* int *max_samples,* ReadCondition *condition* )

This operation takes a collection of data values (with associated SampleInfo), subject to a filter, from the DataReader.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

**See also**

com.toc.coredx.DDS.TDataReader::take(List<T> received_data, List<SampleInfo> sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states) take()

#### 2.26.2.23 ReturnCode_t unregister_instance ( T *instance_data,* InstanceHandle_t *handle* )

Indicates that the writer will no longer be providing updates the specified instance. If **handle** is not HANDLE_NIL, then **handle** must identify a valid instance that has been previously registered or written by this DataWriter. The current time is used as the source timstamp.

#### 2.26.2.24 ReturnCode_t unregister_instance_w_timestamp ( T *instance_data,* InstanceHandle_t *handle,* Time_t *timestamp* )

Indicates that the writer will no longer be providing updates the specified instance. If **handle** is not HANDLE_NIL, then **handle** must identify a valid instance that has been previously registered or written by this DataWriter. The provided **source_timestamp** is used as the source timstamp.

#### 2.26.2.25 ReturnCode_t write ( T *instance_data,* InstanceHandle_t *handle* )

Publishes the provided **instance_data.** If **handle** is HANDLE_NIL, then the handle is determined from the 'key fields' of **instance_data**.

The current time is used as the source timestamp for the published data.

This routine may block if RELIABILITY kind == RELIABLE, RESOURCE_LIMITS are not UNLIMITED, and the local resources are exhausted. The routine will block at most 'max_blocking_time' as configured in the RELIABLITY QoS. If the routine is still unable to handle the data, after blocking, then RETCODE_TIMEOUT is returned.

The routine may return RETCODE_OUT_OF_RESOURCES if it is determined that no amount of blocking will allow the data to be processed.

On success, RETCODE_OK is returned.

#### 2.26.2.26 ReturnCode_t write_w_timestamp ( T *instance_data,* InstanceHandle_t *handle,* Time_t *timestamp* )

Publishes the provided **instance_data.** If **handle** is HANDLE_NIL, then the handle is determined from the 'key fields' of **instance_data.**

The **source_timestamp** is used as the source timestamp for the published data.

This routine may block if RELIABILITY kind == RELIABLE, RESOURCE_LIMITS are not UNLIMITED, and the local resources are exhausted. The routine will block at most 'max_blocking_time' as configured in the RELIABLITY QoS. If the routine is still unable to handle the data, after blocking, then RETCODE_TIMEOUT is returned.

The routine may return RETCODE_OUT_OF_RESOURCES if it is determined that no amount of blocking will allow the data to be processed.

On success, RETCODE_OK is returned.

# Chapter 3

# Example DDS Generated Code

## 3.1 FooDataReader Class Reference

### 3.1.1 Description

The FooDataReader is an example specialized DataReader (generated by the coredx_ddl compiler) for reading data samples of type 'Foo'.

The FooDataReader is an implementation of the base DataReader class that has been extended to support the 'Foo' data type.

**Note:** the FooDataReader is included here for documentation purposes only, and does not really exist in the com.toc.coredx.DDS package.

Inherits DataReaderI< T >.

#### Public Member Functions

- ReturnCode_t get_key_value (Foo key_holder, InstanceHandle_t handle)
- InstanceHandle_t lookup_instance (Foo instance_data)
- ReturnCode_t return_loan (List< Foo > received_data, List< SampleInfo > sample_infos)
- ReturnCode_t read (List< Foo > received_data, List< SampleInfo > sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states)
- ReturnCode_t take (List< Foo > received_data, List< SampleInfo > sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states)
- ReturnCode_t read_w_condition (List< Foo > received_data, List< SampleInfo > sample_infos, int max_samples, ReadCondition condition)
- ReturnCode_t take_w_condition (List< Foo > received_data, List< SampleInfo > sample_infos, int max_samples, ReadCondition condition)
- ReturnCode_t read_next_sample (Foo received_data, SampleInfo sample_info)
- ReturnCode_t take_next_sample (Foo received_data, SampleInfo sample_info)
- ReturnCode_t read_instance (List< Foo > received_data, List< SampleInfo > sample_infos, int max_samples, InstanceHandle_t handle, uint sample_states, uint view_states, uint instance_states)
- ReturnCode_t take_instance (List< Foo > received_data, List< SampleInfo > sample_infos, int max_samples, InstanceHandle_t handle, uint sample_states, uint view_states, uint instance_states)
- ReturnCode_t read_next_instance (List< Foo > received_data, List< SampleInfo > sample_infos, int max_samples, InstanceHandle_t prev_handle, uint sample_states, uint view_states, uint instance_states)

- ReturnCode_t take_next_instance (List< Foo > received_data, List< SampleInfo > sample_infos, int max_samples, InstanceHandle_t prev_handle, uint sample_states, uint view_states, uint instance_states)
- ReturnCode_t read_next_instance_w_condition (List< Foo > received_data, List< SampleInfo > sample_infos, int max_samples, InstanceHandle_t handle, ReadCondition condition)
- ReturnCode_t take_next_instance_w_condition (List< Foo > received_data, List< SampleInfo > sample_infos, int max_samples, InstanceHandle_t handle, ReadCondition condition)

### 3.1.2 Member Function Documentation

#### 3.1.2.1 ReturnCode_t get_key_value ( Foo *key_holder,* InstanceHandle_t *handle* )

This routine will populate the data structure indicated by **key_holder** with the key infomation identified by **handle.**

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

#### 3.1.2.2 InstanceHandle_t lookup_instance ( Foo *instance_data* )

Returns the handle that identifies the data instance provided in **instance_data.** The 'key' field values of the data are associated with a unique handle.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

#### 3.1.2.3 ReturnCode_t read ( List< Foo > *received_data,* List< SampleInfo > *sample_infos,* int *max_samples,* uint *sample_states,* uint *view_states,* uint *instance_states* )

This operation accesses a collection of data values (with associated SampleInfo) within the DataReader. This routine, and the related **take(),** provide the interface for an application to access published data. There are several varieties of **read()** and **take(),** to facilitate different access patterns or approaches.

The primary difference between **read()** and **take()** is that **take()** removes all returned data samples from the DataReader while **read()** does not. Sequential **read()** calls will return the same data samples each time (if nothing else changes); while sequential **take()** calls will return data samples for only the first call. Subsequent **take()** calls will return an empty collection (if no new data arrives).

The specific behavior of **read()** depends on several things: input paramters, the QoS settings of the DataReader, and the state of recieved data. First, the **received_data** and **sample_infos** arguments affect the following:

- how many samples are returned, and
- whether the returned data should be 'loaned' or copied.

The argument **max_samples** is used to further limit the number of samples returned.

The **sample_states, view_states,** and **instance_states** arguments are used to selectively add data samples to the returned collections. These arguments indicate the desired 'states' for data samples and instances. These state arguments are bit masks; they can be the bit-wise OR of several individual state flags or they may use the special 'ANY' constants (e.g.: DDS.ANY_SAMPLE_STATE). Only samples that have a matching state for all three categories are added to the returned collection.

The order of samples in the returned collections is determined by the **PRESENTATION** and **DESTINATION_ORDER** QoS policies.

The returned collection is held in **recieved_data** and **samples_infos.** These two sequences operate together to represent a sequence of pairs (data, SampleInfo). Each data item in **received_data** has a corresponding entry in **sample_infos** that provides associated 'meta-data'. See SampleInfo for a description of this meta-data.

In CoreDX DDS, the returned sequences contain 'loaned' data. This provides zero-copy access to the data, and provides a very efficient data access mechanism. Becuase the data is 'loaned' to the application, the application is required to indicate when it is finished accessing the data. This is accomplished by calling return_loan().

The **read()** operation will set the SampleInfo.sample_state to DDS.READ_SAMPLE_STATE.

The **read()** operation may set the SampleInfo.view_state to DDS.NOT_NEW_VIEW_STATE, if a sample of the most recent generation of the instance is read.

If there is no data found, then the **read()** will return RETCODE_NO_DATA.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

#### 3.1.2.4 ReturnCode_t read_instance ( List< Foo > *received_data,* List< SampleInfo > *sample_infos,* int *max_samples,* InstanceHandle_t *handle,* uint *sample_states,* uint *view_states,* uint *instance_states* )

This operation accesses a collection of data values (with associated SampleInfo), belonging to a particular instance, within the DataReader.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

**See also**

com.toc.coredx.DDS.FooDataReader::read(List<Foo> received_data, List<SampleInfo> sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states) read()

#### 3.1.2.5 ReturnCode_t read_next_instance ( List< Foo > *received_data,* List< SampleInfo > *sample_infos,* int *max_samples,* InstanceHandle_t *prev_handle,* uint *sample_states,* uint *view_states,* uint *instance_states* )

This operation accesses a collection of data values (with associated SampleInfo), instance by instance, within the DataReader. To start,pass HANDLE_NIL for parameter prev_handle. After the last instance (in order) has been taken, this routine will return RETCODE_NO_DATA.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

**See also**

com.toc.coredx.DDS.FooDataReader::read(List<Foo> received_data, List<SampleInfo> sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states) read()

#### 3.1.2.6 ReturnCode_t read_next_instance_w_condition ( List< Foo > *received_data,* List< SampleInfo > *sample_infos,* int *max_samples,* InstanceHandle_t *handle,* ReadCondition *condition* )

This operation accesses a collection of data values (with associated SampleInfo), instance by instance subject to a filter, within the DataReader.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

**See also**

com.toc.coredx.DDS.FooDataReader::read(List<Foo> received_data, List<SampleInfo> sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states) read()

#### 3.1.2.7 ReturnCode_t read_next_sample ( Foo *received_data,* SampleInfo *sample_info* )

This operation accesses a data value (with associated SampleInfo) within the DataReader.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

**See also**

com.toc.coredx.DDS.FooDataReader::read(List<Foo> received_data, List<SampleInfo> sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states) read()

#### 3.1.2.8 ReturnCode_t read_w_condition ( List< Foo > *received_data,* List< SampleInfo > *sample_infos,* int *max_samples,* ReadCondition *condition* )

This operation accesses a collection of data values (with associated SampleInfo), subject to a filter, within the DataReader.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

**See also**

com.toc.coredx.DDS.FooDataReader::read(List<Foo> received_data, List<SampleInfo> sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states) read()

#### 3.1.2.9 ReturnCode_t return_loan ( List< Foo > *received_data,* List< SampleInfo > *sample_infos* )

Returns data and sample_info values to a DataReader. When an application calls DataReader **read()** or **take()** operations, these routines 'loan' data and SampleInfo values to the application. [This is an optimization that avoids extra copies of data.] The appliction must return this 'loaned' data to the DataReader. The return_loan() routine indicates to the DataReader that the application no longer requires access to the data and sample_infos.

A call to **return_loan()** operation must be made only if previous **read()** or **take()** calls 'loaned' data to the application. See read() for a discussion of when data is 'loaned'.

If the **received_data** or **sample_infos** parameters provided do not identify data obtained from DataReader **dr,** then the error RETCODE_PRECONDITION_NOT_MET will be returned.

A DataReader cannot be deleted if any 'loans' are outstanding.

**See also**

com.toc.coredx.DDS.FooDataReader::read(FooSeq received_data, List<SampleInfo> sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states) read()

#### 3.1.2.10 ReturnCode_t take ( List< Foo > *received_data,* List< SampleInfo > *sample_infos,* int *max_samples,* uint *sample_states,* uint *view_states,* uint *instance_states* )

This operation takes a collection of data values (with associated SampleInfo) from the DataReader. This routine, and the related **read()**, provide the interface for an application to access published data. There are several varieties of **read()** and **take()**, to facilitate different access patterns or approaches.

The primary difference between **read()** and **take()** is that **take()** removes all returned data samples from the DataReader while **read()** does not. Sequential **read()** calls will return the same data samples each time (if nothing else changes); while sequential **take()** calls will return data samples for only the first call. Subsequent **take()** calls will return an empty collection (if no new data arrives).

The specific behavior of **take()** depends on several things: input paramters, the QoS settings of the DataReader, and the state of recieved data. First, the **received_data** and **sample_infos** arguments affect the following:

- how many samples are returned, and
- whether the returned data should be 'loaned' or copied.

The argument **max_samples** is used to further limit the number of samples returned.

The **sample_states, view_states,** and **instance_states** arguments are used to selectively add data samples to the returned collections. These arguments indicate the desired 'states' for data samples and instances. These state arguments are bit masks; they can be the bit-wise OR of several individual state flags or they may use the special 'ANY' constants (e.g.: DDS.ANY_SAMPLE_STATE). Only samples that have a matching state for all three categories are added to the returned collection.

The order of samples in the returned collections is determined by the **PRESENTATION** and **DESTINATION_ORDER** QoS policies.

The returned collection is held in **recieved_data** and **samples_infos.** These two sequences operate together to represent a sequence of pairs (data, SampleInfo). Each data item in **received_data** has a corresponding entry in **sample_infos** that provides associated 'meta-data'. See SampleInfo for a description of this meta-data.

In CoreDX DDS, the returned sequences contain 'loaned' data. This provides zero-copy access to the data, and provides a very efficient data access mechanism. Becuase the data is 'loaned' to the application, the application is required to indicate when it is finished accessing the data. This is accomplished by calling return_loan().

The **take()** operation will set the SampleInfo.sample_state to DDS.READ_SAMPLE_STATE.

The **take()** operation may set the SampleInfo.view_state to DDS.NOT_NEW_VIEW_STATE, if a sample of the most recent generation of the instance is taken.

If there is no data found, then the **take()** will return RETCODE_NO_DATA.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

#### 3.1.2.11 ReturnCode_t take_instance ( List< Foo > *received_data,* List< SampleInfo > *sample_infos,* int *max_samples,* InstanceHandle_t *handle,* uint *sample_states,* uint *view_states,* uint *instance_states* )

This operation takes a collection of data values (with associated SampleInfo), belonging to a particular instance, from the DataReader.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

**See also**

com.toc.coredx.DDS.FooDataReader::take(List<Foo> received_data, List<SampleInfo> sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states) take()

#### 3.1.2.12 ReturnCode_t take_next_instance ( List< Foo > *received_data,* List< SampleInfo > *sample_infos,* int *max_samples,* InstanceHandle_t *prev_handle,* uint *sample_states,* uint *view_states,* uint *instance_states* )

This operation takes a collection of data values (with associated SampleInfo), instance by instance, from the DataReader. To start,pass HANDLE_NIL for parameter prev_handle. After the last instance (in order) has been taken, this routine will return RETCODE_NO_DATA.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

**See also**

com.toc.coredx.DDS.FooDataReader::take(List<Foo> received_data, List<SampleInfo> sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states) take()

#### 3.1.2.13 ReturnCode_t take_next_instance_w_condition ( List< Foo > *received_data,* List< SampleInfo > *sample_infos,* int *max_samples,* InstanceHandle_t *handle,* ReadCondition *condition* )

This operation takes a collection of data values (with associated SampleInfo), instance by instance subject to a filter, from the DataReader.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

**See also**

com.toc.coredx.DDS.FooDataReader::take(List<Foo> received_data, List<SampleInfo> sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states) take()

#### 3.1.2.14 ReturnCode_t take_next_sample ( Foo *received_data,* SampleInfo *sample_info* )

This operation takes a data value (with associated SampleInfo) from the DataReader.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

**See also**

com.toc.coredx.DDS.FooDataReader::take(List<Foo> received_data, List<SampleInfo> sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states) take()

#### 3.1.2.15 ReturnCode_t take_w_condition ( List< Foo > *received_data,* List< SampleInfo > *sample_infos,* int *max_samples,* ReadCondition *condition* )

This operation takes a collection of data values (with associated SampleInfo), subject to a filter, from the DataReader.

This routine is data type specific. The generated type specific DataReader includes an implementation of this routine which should be used to support type-safety.

**See also**

com.toc.coredx.DDS.FooDataReader::take(List<Foo> received_data, List<SampleInfo> sample_infos, int max_samples, uint sample_states, uint view_states, uint instance_states) take()

## 3.2 FooDataWriter Class Reference

### 3.2.1 Description

The FooDataWriter is an example specialized DataWriter (generated by the coredx_ddl compiler) for writing data samples of type 'Foo'.

The FooDataWriter is an implementation of the base DataWriter class that has been extended to support the 'Foo' data type.

**Note:** the FooDataWriter is included here for documentation purposes only, and does not really exist in the com.toc.coredx.DDS package.

Inherits DataWriterI< T >.

### Public Member Functions

- ReturnCode_t get_key_value (Foo key_holder, InstanceHandle_t handle)
- InstanceHandle_t lookup_instance (Foo instance_data)
- InstanceHandle_t register_instance (Foo instance_data)
- InstanceHandle_t register_instance_w_timestamp (Foo instance_data, Time_t ts)
- ReturnCode_t unregister_instance (Foo instance_data, InstanceHandle_t handle)
- ReturnCode_t unregister_instance_w_timestamp (Foo instance_data, InstanceHandle_t handle, Time_t timestamp)
- ReturnCode_t write (Foo instance_data, InstanceHandle_t handle)
- ReturnCode_t write_w_timestamp (Foo instance_data, InstanceHandle_t handle, Time_t timestamp)
- ReturnCode_t dispose (Foo instance_data, InstanceHandle_t handle)
- ReturnCode_t dispose_w_timestamp (Foo instance_data, InstanceHandle_t handle, Time_t timestamp)

### 3.2.2 Member Function Documentation

#### 3.2.2.1 ReturnCode_t dispose ( Foo *instance_data,* InstanceHandle_t *handle* )

Indicates that the instance no longer exists. If **handle** is not HANDLE_NIL, then **handle** must identify a valid instance that has been previously registered or written by this DataWriter. The current time is used as the source timstamp.

#### 3.2.2.2 ReturnCode_t dispose_w_timestamp ( Foo *instance_data,* InstanceHandle_t *handle,* Time_t *timestamp* )

Indicates that the instance no longer exists. If **handle** is not HANDLE_NIL, then **handle** must identify a valid instance that has been previously registered or written by this DataWriter. The **source_timestamp** is used as the source timstamp for the published message.

#### 3.2.2.3 ReturnCode_t get_key_value ( Foo *key_holder,* InstanceHandle_t *handle* )

This routine will populate the data structure indicated by **key_holder** with the key infomation identified by **handle.**

This routine is data type specific. The generated type specific DataWriter includes an implementation of this routine which should be used to support type-safety.

#### 3.2.2.4 InstanceHandle_t lookup_instance ( Foo *instance_data* )

Returns the handle that identifies the data instance provided in **instance_data.** The 'key' field values of the data are associated with a unique handle.

#### 3.2.2.5 InstanceHandle_t register_instance ( Foo *instance_data* )

Declares the existance of an instance identified by the 'key fields' in **instance_data.** The handle that uniquely identifies the instance is returned. The current time is used as the source timestamp.

#### 3.2.2.6 InstanceHandle_t register_instance_w_timestamp ( Foo *instance_data,* Time_t *ts* )

Declares the existance of an instance identified by the 'key fields' in **instance_data.** The handle that uniquely identifies the instance is returned. The **source_timestamp** is used as the source timestamp.

#### 3.2.2.7 ReturnCode_t unregister_instance ( Foo *instance_data,* InstanceHandle_t *handle* )

Indicates that the writer will no longer be providing updates the specified instance. If **handle** is not HANDLE_NIL, then **handle** must identify a valid instance that has been previously registered or written by this DataWriter. The current time is used as the source timstamp.

#### 3.2.2.8 ReturnCode_t unregister_instance_w_timestamp ( Foo *instance_data,* InstanceHandle_t *handle,* Time_t *timestamp* )

Indicates that the writer will no longer be providing updates the specified instance. If **handle** is not HANDLE_NIL, then **handle** must identify a valid instance that has been previously registered or written by this DataWriter. The provided **source_timestamp** is used as the source timstamp.

#### 3.2.2.9 ReturnCode_t write ( Foo *instance_data,* InstanceHandle_t *handle* )

Publishes the provided **instance_data.** If **handle** is HANDLE_NIL, then the handle is determined from the 'key fields' of **instance_data**.

The current time is used as the source timestamp for the published data.

This routine may block if RELIABILITY kind == RELIABLE, RESOURCE_LIMITS are not UNLIMITED, and the local resources are exhausted. The routine will block at most 'max_blocking_time' as configured in the RELIABLITY QoS. If the routine is still unable to handle the data, after blocking, then RETCODE_TIMEOUT is returned.

The routine may return RETCODE_OUT_OF_RESOURCES if it is determined that no amount of blocking will allow the data to be processed.

On success, RETCODE_OK is returned.

#### 3.2.2.10 ReturnCode_t write_w_timestamp ( Foo *instance_data,* InstanceHandle_t *handle,* Time_t *timestamp* )

Publishes the provided **instance_data.** If **handle** is HANDLE_NIL, then the handle is determined from the 'key fields' of **instance_data.**

The **source_timestamp** is used as the source timestamp for the published data.

This routine may block if RELIABILITY kind == RELIABLE, RESOURCE_LIMITS are not UNLIMITED, and the local resources are exhausted. The routine will block at most 'max_blocking_time' as configured in the RELIABLITY QoS. If the routine is still unable to handle the data, after blocking, then RETCODE_TIMEOUT is returned.

The routine may return RETCODE_OUT_OF_RESOURCES if it is determined that no amount of blocking will allow the data to be processed.

On success, RETCODE_OK is returned.

# Chapter 4

# Entity QoS Collections

## 4.1 DataReaderQos Class Reference

### 4.1.1 Description

Structure that holds DataReader Quality of Service policies.

**See also**

DataReader::set_qos()
DataReader::get_qos()
Subscriber::create_datareader()
Subscriber::set_default_datareader_qos()
Subscriber::get_default_datareader_qos()

### Public Attributes

- DurabilityQosPolicy durability
- DeadlineQosPolicy deadline
- LatencyBudgetQosPolicy latency_budget
- LivelinessQosPolicy liveliness
- ReliabilityQosPolicy reliability
- DestinationOrderQosPolicy destination_order
- HistoryQosPolicy history
- ResourceLimitsQosPolicy resource_limits
- UserDataQosPolicy user_data
- OwnershipQosPolicy ownership
- TimeBasedFilterQosPolicy time_based_filter
- ReaderDataLifecycleQosPolicy reader_data_lifecycle
- DataRepresentationQosPolicy representation
- TypeConsistencyEnforcementQosPolicy type_consistency
- RpcQosPolicy rpc
- EntityNameQosPolicy entity_name
- LoggingQosPolicy logging
- RTPSReaderQosPolicy rtps_reader

### 4.1.2 Member Data Documentation

#### 4.1.2.1 DeadlineQosPolicy deadline

The requested update frequency for data instances.

#### 4.1.2.2 DestinationOrderQosPolicy destination_order

The destination order logic requested by the DataReader.

#### 4.1.2.3 DurabilityQosPolicy durability

The durability policy requested by the DataReader.

#### 4.1.2.4 EntityNameQosPolicy entity_name

entity_name

#### 4.1.2.5 HistoryQosPolicy history

The data history requested by the DataReader.

#### 4.1.2.6 LatencyBudgetQosPolicy latency_budget

The latency requested by the DataReader.

#### 4.1.2.7 LivelinessQosPolicy liveliness

The liveliness mechanism requested by the DataReader.

#### 4.1.2.8 LoggingQosPolicy logging

logging

#### 4.1.2.9 OwnershipQosPolicy ownership

The type of 'ownership' offered by the DataReader.

#### 4.1.2.10 ReaderDataLifecycleQosPolicy reader_data_lifecycle

Controls the auto-purge behavior of the DataReader.

#### 4.1.2.11 ReliabilityQosPolicy reliability

The transport reliability requested by the DataReader.

#### 4.1.2.12 DataRepresentationQosPolicy representation

Informs DataWriter(s) of the data representation(s) supported.

#### 4.1.2.13 ResourceLimitsQosPolicy resource_limits

The resource limits set on the DataReader.

#### 4.1.2.14 RpcQosPolicy rpc

Configure RPC relevant settings: instance_name, related_entity,and topic_aliases.

#### 4.1.2.15 RTPSReaderQosPolicy rtps_reader

rtps reader configuration

#### 4.1.2.16 TimeBasedFilterQosPolicy time_based_filter

The maximum update frequency required/desired by the DataReader.

#### 4.1.2.17 TypeConsistencyEnforcementQosPolicy type_consistency

Influences the algorithm that matches DataReaders and DataWriters based on their data type compatibility.

#### 4.1.2.18 UserDataQosPolicy user_data

A sequence of octets associated with the DataReader.

## 4.2 DataWriterQos Class Reference

### 4.2.1 Description

Structure that holds DataWriter Quality of Service policies.

**See also**

DataWriter::set_qos()
DataWriter::get_qos()
Publisher::create_datawriter()
Publisher::set_default_datawriter_qos()
Publisher::get_default_datawriter_qos()

### Public Attributes

- DurabilityQosPolicy durability
- DurabilityServiceQosPolicy durability_service
- DeadlineQosPolicy deadline
- LatencyBudgetQosPolicy latency_budget
- LivelinessQosPolicy liveliness
- ReliabilityQosPolicy reliability
- DestinationOrderQosPolicy destination_order
- HistoryQosPolicy history
- ResourceLimitsQosPolicy resource_limits
- TransportPriorityQosPolicy transport_priority
- LifespanQosPolicy lifespan
- UserDataQosPolicy user_data
- OwnershipQosPolicy ownership
- OwnershipStrengthQosPolicy ownership_strength
- WriterDataLifecycleQosPolicy writer_data_lifecycle
- DataRepresentationQosPolicy representation
- RpcQosPolicy rpc
- EntityNameQosPolicy entity_name
- LoggingQosPolicy logging
- RTPSWriterQosPolicy rtps_writer

### 4.2.2 Member Data Documentation

#### 4.2.2.1 DeadlineQosPolicy deadline

The deadline committed to by the DataWriter.

#### 4.2.2.2 DestinationOrderQosPolicy destination_order

The destination order logic offered by the DataWriter.

#### 4.2.2.3 DurabilityQosPolicy durability

The durability policy offered by the DataWriter.

#### 4.2.2.4 DurabilityServiceQosPolicy durability_service

The durability service configuration offered by the DataWriter.

#### 4.2.2.5 EntityNameQosPolicy entity_name

entity_name

#### 4.2.2.6 HistoryQosPolicy history

The data history maintained by the DataWriter.

#### 4.2.2.7 LatencyBudgetQosPolicy latency_budget

The latency allowed by the DataWriter.

#### 4.2.2.8 LifespanQosPolicy lifespan

The expiration time for old samples managed by the DataWriter.

#### 4.2.2.9 LivelinessQosPolicy liveliness

The liveliness mechanism offered by the DataWriter.

#### 4.2.2.10 LoggingQosPolicy logging

logging

#### 4.2.2.11 OwnershipQosPolicy ownership

The type of 'ownership' offered by the DataWriter.

#### 4.2.2.12 OwnershipStrengthQosPolicy ownership_strength

The measure of 'ownership strength' offered by the DataWriter.

#### 4.2.2.13 ReliabilityQosPolicy reliability

The transport reliability offered by the DataWriter.

#### 4.2.2.14 DataRepresentationQosPolicy representation

Informs DataReader(s) of the data representation(s) supported.

#### 4.2.2.15 ResourceLimitsQosPolicy resource_limits

The resource limits set on the DataWriter.

#### 4.2.2.16 RpcQosPolicy rpc

Configure RPC relevant settings: instance_name, related_entity,and topic_aliases.

#### 4.2.2.17 RTPSWriterQosPolicy rtps_writer

rtps writer configuration

#### 4.2.2.18 TransportPriorityQosPolicy transport_priority

The transport priority supported by the DataWriter.

#### 4.2.2.19 UserDataQosPolicy user_data

A sequence of octets associated with the DataWriter.

#### 4.2.2.20 WriterDataLifecycleQosPolicy writer_data_lifecycle

Indicates if unregistered instances should be automatically disposed by the DataWriter.

## 4.3 DomainParticipantFactoryQos Class Reference

### 4.3.1 Description

Structure that holds DomainParticipantFactory Quality of Service policies.

**See also**

DomainParticipantFactory::set_qos()
DomainParticipantFactory::get_qos()

### Public Attributes

- EntityFactoryQosPolicy entity_factory

### 4.3.2 Member Data Documentation

#### 4.3.2.1 EntityFactoryQosPolicy entity_factory

Controls the behavior of the DomainParticipant 'create' operations.

## 4.4 DomainParticipantQos Class Reference

### 4.4.1 Description

Structure that holds DomainParticipant Quality of Service policies.

**See also**

DomainParticipant::set_qos()
DomainParticipant::get_qos()
DomainParticipantFactory::create_participant()
DomainParticipantFactory::set_default_participant_qos()
DomainParticipantFactory::get_default_participant_qos()

### Public Attributes

- UserDataQosPolicy user_data
- EntityFactoryQosPolicy entity_factory
- PropertyQosPolicy properties
- EntityNameQosPolicy entity_name
- LoggingQosPolicy logging
- PeerParticipantQosPolicy peer_participants
- DiscoveryQosPolicy discovery
- ThreadModelQosPolicy thread_model

### 4.4.2 Member Data Documentation

#### 4.4.2.1 DiscoveryQosPolicy discovery

Configure the behavior of discovery and discovery entities

#### 4.4.2.2 EntityFactoryQosPolicy entity_factory

Controls the behavior of the DomainParticipant **create** operations.

#### 4.4.2.3 EntityNameQosPolicy entity_name

Assign a name to this entity

#### 4.4.2.4 LoggingQosPolicy logging

Control the logging output

#### 4.4.2.5 PeerParticipantQosPolicy peer_participants

Specifies a list of DomainParticipant peers to attempt communication with. Augments default discovery.

#### 4.4.2.6 PropertyQosPolicy properties

Additional name:value pair properties (if propagate=true, included in discovery)

#### 4.4.2.7 ThreadModelQosPolicy thread_model

Configure the threading features of CoreDX DDS

#### 4.4.2.8 UserDataQosPolicy user_data

A sequence of octets associated with a DomainParticipant.

## 4.5 PublisherQos Class Reference

### 4.5.1 Description

Structure that holds Publisher Quality of Service policies.

**See also**

Publisher::set_qos()
Publisher::get_qos()
DomainParticipant::create_publisher()
DomainParticipant::set_default_publisher_qos()
DomainParticipant::get_default_publisher_qos()

### Public Attributes

- PresentationQosPolicy presentation
- PartitionQosPolicy partition
- GroupDataQosPolicy group_data
- EntityFactoryQosPolicy entity_factory
- EntityNameQosPolicy entity_name
- LoggingQosPolicy logging

### 4.5.2 Member Data Documentation

#### 4.5.2.1 EntityFactoryQosPolicy entity_factory

Controls the behavior of the Publisher_create_datawriter() operation.

#### 4.5.2.2 EntityNameQosPolicy entity_name

entity_name

#### 4.5.2.3 GroupDataQosPolicy group_data

A sequence of octets associated with the Publisher.

#### 4.5.2.4 LoggingQosPolicy logging

logging

#### 4.5.2.5 PartitionQosPolicy partition

Establishes a logical data partition. DataWriters and DataReaders that are 'in' the same partition (ie, the partition of the containing Publisher and Subscriber match) can communicate. If the partitions do not match, then they cannot communicate.

#### 4.5.2.6 PresentationQosPolicy presentation

Controls the presentation of groups of changes.

**See also**

Publisher::begin_coherent_changes() begin_coherent_changes()
Publisher::end_coherent_changes() end_coherent_changes()
Subscriber::begin_access() begin_access()
Subscriber::end_access() end_access()

## 4.6 SubscriberQos Class Reference

### 4.6.1 Description

Structure that holds DDS_Subscriber Quality of Service policies.

**See also**

Subscriber::set_qos(SubscriberQos)
Subscriber::get_qos(SubscriberQos)
DomainParticipant::create_subscriber()
DomainParticipant::set_default_subscriber_qos()
DomainParticipant::get_default_subscriber_qos()

### Public Attributes

- PresentationQosPolicy presentation
- PartitionQosPolicy partition
- GroupDataQosPolicy group_data
- EntityFactoryQosPolicy entity_factory
- EntityNameQosPolicy entity_name
- LoggingQosPolicy logging

### 4.6.2 Member Data Documentation

#### 4.6.2.1 EntityFactoryQosPolicy entity_factory

Controls the behavior of the Subscriber.create_datareader() operation.

#### 4.6.2.2 EntityNameQosPolicy entity_name

entity_name

#### 4.6.2.3 GroupDataQosPolicy group_data

A sequence of octets associated with the Publisher.

#### 4.6.2.4 LoggingQosPolicy logging

logging

#### 4.6.2.5 PartitionQosPolicy partition

- Establishes a logical data partition. DataWriters and DataReaders that are 'in' the same partition (ie, the partition of the containing Publisher and Subscriber match) can communicate. If the partitions do not match, then they cannot communicate.

#### 4.6.2.6 PresentationQosPolicy presentation

Controls the presentation of groups of changes.

**See also**

Publisher::begin_coherent_changes()
Publisher::end_coherent_changes()
Subscriber::begin_access()
Subscriber::end_access()

## 4.7 TopicQos Class Reference

### 4.7.1 Description

Structure that holds DDS_Topic Quality of Service policies.

**See also**

Topic::set_qos()
Topic::get_qos()
DomainParticipant::create_topic()
DomainParticipant::set_default_topic_qos()

### Public Attributes

- TopicDataQosPolicy topic_data
- DurabilityQosPolicy durability
- DurabilityServiceQosPolicy durability_service
- DeadlineQosPolicy deadline
- LatencyBudgetQosPolicy latency_budget
- LivelinessQosPolicy liveliness
- ReliabilityQosPolicy reliability
- DestinationOrderQosPolicy destination_order
- HistoryQosPolicy history
- ResourceLimitsQosPolicy resource_limits
- TransportPriorityQosPolicy transport_priority
- LifespanQosPolicy lifespan
- OwnershipQosPolicy ownership
- EntityNameQosPolicy entity_name
- LoggingQosPolicy logging

### 4.7.2 Member Data Documentation

#### 4.7.2.1 DeadlineQosPolicy deadline

deadline

#### 4.7.2.2 DestinationOrderQosPolicy destination_order

destination_order

#### 4.7.2.3 DurabilityQosPolicy durability

durability

#### 4.7.2.4 DurabilityServiceQosPolicy durability_service

durability_service

#### 4.7.2.5 EntityNameQosPolicy entity_name

entity_name

#### 4.7.2.6 HistoryQosPolicy history

history

#### 4.7.2.7 LatencyBudgetQosPolicy latency_budget

latency_budget

#### 4.7.2.8 LifespanQosPolicy lifespan

lifespan

#### 4.7.2.9 LivelinessQosPolicy liveliness

liveliness

#### 4.7.2.10 LoggingQosPolicy logging

logging

#### 4.7.2.11 OwnershipQosPolicy ownership

ownership

#### 4.7.2.12 ReliabilityQosPolicy reliability

reliability

#### 4.7.2.13 ResourceLimitsQosPolicy resource_limits

resource_limits

#### 4.7.2.14 TopicDataQosPolicy topic_data

topic_data

#### 4.7.2.15 TransportPriorityQosPolicy transport_priority

transport_priority

# Chapter 5

# QoS Policies

## 5.1 DataRepresentationQosPolicy Struct Reference

### 5.1.1 Description

This QoS policy establishes the DataRepresentation (data encoding) offered by a DataWriter or requested by a DataReader.

DataWriters offer a specific single encoding. DataReaders request zero or more encodings.

The policy is compatible between a Writer and a Reader if there is a matching encoding between offered and requested.

#### Public Attributes

- ushort[] value

### 5.1.2 Member Data Documentation

#### 5.1.2.1 ushort [] value

the set of representation kinds supported

## 5.2 DeadlineQosPolicy Struct Reference

### 5.2.1 Description

This QoS policy establishes a minimum update period for data instances.

DataWriters specify that each data instance will be updated at least once every 'period'. A DataReader requests that the Writer commit to updating each instance at least once every 'period'.

The policy is compatible (between a Writer and a Reader) if the offered deadline period <= requested deadline period.

If a DataWriter fails to meet the update period, then the DataWriter is notified by a call to the offered_deadline_missed listener, and the DataReader is notified by a call to the requested_deadlin_missed listener.

**See also**

DataReaderListener::on_requested_deadline_missed
DataWriterListener::on_offered_deadline_missed

### Public Attributes

- Duration_t period

### 5.2.2 Member Data Documentation

#### 5.2.2.1 Duration_t period

period

## 5.3 DestinationOrderQosPolicy Struct Reference

### 5.3.1 Description

This QoS policy controls how each Subscriber orders received data samples.

Can be **BY_RECEPTION_TIMESTAMP** or **BY_SOURCE_TIMESTAMP**. The samples are ordered either based on their order of reception, or by the order of timestamps generated by the source (DataWriter).

CoreDX DDS currently implements only BY_RECEPTION_TIMESTAMP.

### Public Attributes

- DestinationOrderQosPolicyKind kind

### 5.3.2 Member Data Documentation

#### 5.3.2.1 DestinationOrderQosPolicyKind kind

ordering

## 5.4 DiscoveryQosPolicy Struct Reference

### 5.4.1 Description

QoS Policy for configuring aspects of the Discovery and Builtin entities

### Public Attributes

- DiscoveryQosPolicyDiscoveryKind discovery_kind
- int guid_pid
- Duration_t dpd_push_period
- Duration_t dpd_lease_duration
- Duration_t heartbeat_period
- Duration_t nack_response_delay
- Duration_t nack_suppress_delay
- uint min_buffer_size
- uint max_buffer_size
- Duration_t heartbeat_response_delay
- byte send_initial_nack
- byte send_msglen_submsg

### 5.4.2 Member Data Documentation

#### 5.4.2.1 DiscoveryQosPolicyDiscoveryKind discovery_kind

discovery kind

#### 5.4.2.2 Duration_t dpd_lease_duration

How long a discoverd Participant is considered alive without hearing from them

#### 5.4.2.3 Duration_t dpd_push_period

send DiscoveredParticipantData each period

#### 5.4.2.4 int guid_pid

override the default pid field in the GUID

#### 5.4.2.5 Duration_t heartbeat_period

The heartbeat_period to configure for builtin writers

#### 5.4.2.6 Duration_t heartbeat_response_delay

The heartbeat_response_delay to configure for builtin readers

#### 5.4.2.7 uint max_buffer_size

maximum tx buffer size in bytes for builtin writers

#### 5.4.2.8 uint min_buffer_size

minimum tx buffer size in bytes for builtin writers

#### 5.4.2.9 Duration_t nack_response_delay

The nack_response_delay to configure for builtin writers

#### 5.4.2.10 Duration_t nack_suppress_delay

The nack_suppress_delay to configure for builtin writers

#### 5.4.2.11 byte send_initial_nack

send a ACKNACK immediately after matching with new Writer.

#### 5.4.2.12 byte send_msglen_submsg

include 'msglen' submessage in RTPS header

## 5.5 DurabilityQosPolicy Struct Reference

### 5.5.1 Description

The DurablityQosPolicy controls the durablity of data. The DDS API identifies several degrees of data durability.

1. VOLATILE: The data is volatile, and is not persisted beyond the initial publication.
2. TRANSIENT_LOCAL: The data is persisted locally within the source DataWriter. If that DataWriter is destroyed, or the containing application exits, the data is no longer available.
3. TRANSIENT: The data is persisted beyond the lifecycle of the originating DataWriter, and is available even after that DataWriter has been destroyed. However, data does not persist a reboot of the computer.
4. PERSISTENT: The data is persisted to 'off-line' storage, and is available even after a system reboot.

A DataWriter can offer to provide a level of durablity for data that it generates, and a DataReader requests the level of durability that it requires. If the DataReader requests a 'higher' level of durability than that offered by the DataWriter, then the QoS policy is incompatible.

CoreDX DDS currently supports VOLATILE and TRANSIENT_LOCAL.

#### Public Attributes

- DurabilityQosPolicyKind kind

### 5.5.2 Member Data Documentation

#### 5.5.2.1 DurabilityQosPolicyKind kind

durability kind

## 5.6 DurabilityServiceQosPolicy Struct Reference

### 5.6.1 Description

The DurablityServiceQosPolicy supports the durablity of data. The DDS API identifies several degrees of data durability.

1. VOLATILE: The data is volatile, and is not persisted beyond the initial publication.
2. TRANSIENT_LOCAL: The data is persisted locally within the source DataWriter. If that DataWriter is destroyed, or the containing application exits, the data is no longer available.
3. TRANSIENT: The data is persisted beyond the lifecycle of the originating DataWriter, and is available even after that DataWriter has been destroyed. However, data does not persist a reboot of the computer.
4. PERSISTENT: The data is persisted to 'off-line' storage, and is available even after a system reboot.

If a durability of TRANSIENT or PERSISTENT is specified, then a service beyond the source DataWriter must be established to provide advanced data durability. This QoS policy helps to configure that durablity service.

CoreDX DDS currently supports VOLATILE and TRANSIENT_LOCAL.

### Public Attributes

- Duration_t service_cleanup_delay
- HistoryQosPolicyKind history_kind
- int history_depth
- int max_samples
- int max_instances
- int max_samples_per_instance

### 5.6.2 Member Data Documentation

#### 5.6.2.1 int history_depth

history depth

#### 5.6.2.2 HistoryQosPolicyKind history_kind

history kind

#### 5.6.2.3 int max_instances

max instances

#### 5.6.2.4 int max_samples

max samples

#### 5.6.2.5 int max_samples_per_instance

max samples per instance

#### 5.6.2.6 Duration_t service_cleanup_delay

cleanup delay

## 5.7 EntityFactoryQosPolicy Struct Reference

### 5.7.1 Description

Controls the behavior of entity factories. Several DDS entities act as factories to create, control, and destroy other DDS entities. For example, the DomainParticipant is a factory for Publisher, Subscriber, and Topic entities. This QoS controls the behavior of a factory.

A DDS entity can be enabled at creation time. If the autoenable_created_entities is set to **true**, then the factory will automatically call enable() on the entity during creation. Otherwise, the entity is created but not enabled. The application must specifically call enable() on the returned entity.

### Public Attributes

- bool autoenable_created_entities

### 5.7.2 Member Data Documentation

#### 5.7.2.1 bool autoenable_created_entities

automtically enable entites created by this factory

## 5.8 EntityNameQosPolicy Struct Reference

### 5.8.1 Description

Assign an arbitrary name to a DDS entity.

#### Public Attributes

- char[ ] value

### 5.8.2 Member Data Documentation

#### 5.8.2.1 char [ ] value

value

## 5.9 GroupDataQosPolicy Struct Reference

### 5.9.1 Description

Allows the application to attach arbitrary information to a Publisher or Subscriber.

The value of the GROUP_DATA is propagated and made available to applications through the built-in topics.

The group data is implemented as a vector of Bytes, and can be used to store any application defined data.

This QoS can be used by an application in combination with the DataReaderListener and DataWriterListener to implement conditional matching policies.

### Public Attributes

- byte[] value

### 5.9.2 Member Data Documentation

#### 5.9.2.1 byte [] value

group data

## 5.10 HistoryQosPolicy Struct Reference

### 5.10.1 Description

Controls the ammount of historical data maintained by a DataReader or DataWriter.

The history kind can be set to KEEP_ALL_HISTORY_QOS or KEEP_LAST_HISTORY_QOS.

If the history kind is KEEP_LAST, then the history depth field is used to determine the number of samples to keep. The default setting is KEEP_LAST with a depth of 1. This means that for a single sample (the most recent) is kept for each data instance.

If the history kind is KEEP_ALL, then the depth field is unused, and all samples will be kept, within the limits set by the ResourceLimits QoS policy.

This QoS policy must be consistent with the ResourceLimits policy.

**See also**

com.toc.coredx.DDS.ResourceLimitsQosPolicy

### Public Attributes

- HistoryQosPolicyKind kind
- int depth

### 5.10.2 Member Data Documentation

#### 5.10.2.1 int depth

history depth

#### 5.10.2.2 HistoryQosPolicyKind kind

history kind

## 5.11 LatencyBudgetQosPolicy Struct Reference

### 5.11.1 Description

Specifies allowable latency.

The latency budget is used internally by CoreDX DDS to optimize data messaging. If the middleware is provided a latency budget that is non-zero, then several internal operations can be optimized to reduce message and processing overhead.

To be compatible, the DataWriter must offer a latency_budget that is equal to or smaller than that requested by the DataReader.

### Public Attributes

- Duration_t duration

### 5.11.2 Member Data Documentation

#### 5.11.2.1 Duration_t duration

duration

## 5.12 LifespanQosPolicy Struct Reference

### 5.12.1 Description

Specifies the maximum duration of validity of the data written by the DataWriter.

The default value of the lifespan duration is infinite.

### Public Attributes

- Duration_t duration

### 5.12.2 Member Data Documentation

#### 5.12.2.1 Duration_t duration

duration

## 5.13 LivelinessQosPolicy Struct Reference

### 5.13.1 Description

Determines the mechanism and parameters used by the application to determine whether an Entity is alive.

The QoS defines the kind of liveliness (how liveliness is maintained) as well as a 'lease_duration' which specifies an interval for checking liveliness. The liveliness kind can be one of the following:

- AUTOMATIC
- MANUAL_BY_PARTICIPANT
- MANUAL_BY_TOPIC

The lease_duration indicates the maximum interval between liveliness indications from the publishing entity. If this period elapses with no indication from the publishing entity, then the entity is considered not alive.

Changes in liveliness are indicated to the application through status changes and the DataWriterListener.on_liveliness_lost and DataReaderListener.on_liveliness_changed listeners.

**See also**

DataReaderListener::on_liveliness_changed
DataWriterListener::on_liveliness_lost

### Public Attributes

- LivelinessQosPolicyKind kind
- Duration_t lease_duration

### 5.13.2 Member Data Documentation

#### 5.13.2.1 LivelinessQosPolicyKind kind

liveliness kind

#### 5.13.2.2 Duration_t lease_duration

lease duration

## 5.14 LoggingQosPolicy Struct Reference

### 5.14.1 Description

Controls the amount and kind of information that is logged

The application must be linked with the instrumented library in order to enable logging with this LoggingQosPolicy.

The level of log output is controlled by a series of bitmapped flags. Each class of log message is enabled by setting a flag to 1, and is disabled by 0.

### Public Attributes

- uint flags

### 5.14.2 Member Data Documentation

#### 5.14.2.1 uint flags

flags see LoggingFlagsKind

## 5.15 OwnershipQosPolicy Struct Reference

### 5.15.1 Description

Determines instance ownership in the case of multple writers. CoreDX DDS supports both SHARED_OWNERSHIP_QOS and EXCLUSIVE_OWNERSHIP_QOS.

### Public Attributes

- OwnershipQosPolicyKind kind

### 5.15.2 Member Data Documentation

#### 5.15.2.1 OwnershipQosPolicyKind kind

ownership kind

## 5.16 OwnershipStrengthQosPolicy Struct Reference

### 5.16.1 Description

Defines the strength, or priority, of a Writer. The strength is used to determine ownership in the case of EXCLUSIVE_OWNERSHIP_QOS. When multiple writers publish data about the same instance, the **stronger** writer is considered the owner, and data from other writers is not delivered to the reader.

### Public Attributes

- int value

### 5.16.2 Member Data Documentation

#### 5.16.2.1 int value

strength

## 5.17 PartitionQosPolicy Struct Reference

### 5.17.1 Description

Defines a logical data partition. This QoS is assigned to a Publisher or Subscriber. DataWriters and DataReaders exist within the partition defined by their parent. DataWriters and DataReaders communicate only if they have a matching partition (and all other QoS and topic parameters match).

The partition QoS can be changed dynamically. Dynamic changes to partition may cause new matches between DataWriters and DataReaders or may break existing matches.

The partition QoS comprises a vector of Strings. Two Partitions 'match' if any string in one matches any string in the other.

A partition string may be a regular expression with wildcards as defined by POSIX fnmatch API (1003.2-1992 section B.6).

An empty Partition, that is a Partition with an empty name vector, is treated as a Partition with a single empty string of "".

### Public Attributes

- string[ ] name

### 5.17.2 Member Data Documentation

#### 5.17.2.1 string [ ] name

partition names

## 5.18 PeerParticipantQosPolicy Struct Reference

### 5.18.1 Description

Configures a list of DomainParticipant peers to attempt communication with.

If the value list is empty, then default Discovery is used.

### Public Attributes

- List< ParticipantLocator > value
- bool strict_match

### 5.18.2 Member Data Documentation

#### 5.18.2.1 bool strict_match

restrict matching to those peers listed (default is 0/FALSE)

#### 5.18.2.2 List<ParticipantLocator> value

A list of pre-configured peers with which we should attempt communication. Augments default discovery.

## 5.19 PresentationQosPolicy Struct Reference

### 5.19.1 Description

Controls the presentation of received data samples to the application. CoreDX DDS currently supports only the access_scope = INSTANCE_PRESENTATION_QOS policy.

### Public Attributes

- PresentationQosPolicyAccessScopeKind access_scope
- bool coherent_access
- bool ordered_access

### 5.19.2 Member Data Documentation

#### 5.19.2.1 PresentationQosPolicyAccessScopeKind access_scope

The 'scope' of the presentation policy. Determines the extent to which the sample 'coherency' and 'order' are maintained.

#### 5.19.2.2 bool coherent_access

Determines if coherent access is supported within the defined 'scope'.

#### 5.19.2.3 bool ordered_access

Determines if ordered access is supported within the defined 'scope'.

## 5.20 PropertyQosPolicy Struct Reference

### 5.20.1 Description

The PropertyQosPolicy supports adding arbitrary name-value pairs.

This can be used to configure non-standard properties in a portable fashion. In particular, this can be used to configure aspects of Security Plugin implementations.

### Public Attributes

- List< Property_t > value

### 5.20.2 Member Data Documentation

#### 5.20.2.1 List<Property_t> value

collection of Property's <name,value> pairs with flag to indicate propagation via discovery

## 5.21 ReaderDataLifecycleQosPolicy Struct Reference

### 5.21.1 Description

Specifies the lifecycle behavior of data instances managed by the DataReader.

**autopurge_nowriter_samples_delay** indicates how long the DataReader will maintain instance samples that have instance_state NOT_ALIVE_NO_WRITERS.

**autopurge_disposed_samples_delay** indicates how long the DataReader will retain information about instances that have instance_state NOT_ALIVE_DISPOSED.

### Public Attributes

- Duration_t autopurge_nowriter_samples_delay
- Duration_t autopurge_disposed_samples_delay

### 5.21.2 Member Data Documentation

#### 5.21.2.1 Duration_t autopurge_disposed_samples_delay

delay before purging samples from instances that are not alive: disposed

#### 5.21.2.2 Duration_t autopurge_nowriter_samples_delay

delay before purging samples from instances that are not alive: no writers

## 5.22 ReliabilityQosPolicy Struct Reference

### 5.22.1 Description

Indicates the level of reliability offered/provided by the Entity. If kind is RELIABLE_RELIABILITY_QOS, then the middleware will attempt to deliver all samples in the history cache. If samples are not received, then they will be retried.

A kind of BEST_EFFORT_RELIABILITY_QOS indicates that the middleware does not need to retry delivery of data samples.

The samples available in the history cache are affected by the HistoryQosPolicy, ResourceLimitsQosPolicy, and the DurabilityQosPolicy.

Note: If samples are removed from the history cache, then they are not be available to be resent.

**See also**

com.toc.coredx.DDS.HistoryQosPolicy
com.toc.coredx.DDS.ResourceLimitsQosPolicy
com.toc.coredx.DDS.DurabilityQosPolicy

### Public Attributes

- ReliabilityQosPolicyKind kind
- Duration_t max_blocking_time

### 5.22.2 Member Data Documentation

#### 5.22.2.1 ReliabilityQosPolicyKind kind

kind

#### 5.22.2.2 Duration_t max_blocking_time

max time to block before timemout

## 5.23 ResourceLimitsQosPolicy Struct Reference

### 5.23.1 Description

Specifies the resources that the Service can use to maintain data samples and instances.

**See also**

com.toc.coredx.DDS.ReliabilityQosPolicy
com.toc.coredx.DDS.HistoryQosPolicy

### Public Attributes

- int max_samples
- int max_instances
- int max_samples_per_instance
- bool preallocate_samples
- bool preallocate_instances

### 5.23.2 Member Data Documentation

#### 5.23.2.1 int max_instances

maximum instances allowed in the cache

#### 5.23.2.2 int max_samples

maximum samples allowed in the data cache

#### 5.23.2.3 int max_samples_per_instance

maximum samples per instance allowed in the cache

#### 5.23.2.4 bool preallocate_instances

Request that the Reader or Writer pre-allocate the specified number of instances (if max_instances is not unlimited). CoreDX DDS Extension

#### 5.23.2.5 bool preallocate_samples

Request that the Reader or Writer pre-allocate the specified number of samples (if max_samples is not unlimited). CoreDX DDS Extension

## 5.24 RpcQosPolicy Struct Reference

### 5.24.1 Description

Augment a DataWriter or DataReader with RPC specific information.

### Public Attributes

- String service_instance_name
- GUID_t related_entity_guid
- string[ ] topic_aliases

### 5.24.2 Member Data Documentation

#### 5.24.2.1 GUID_t related_entity_guid

guid of the related entity (request / reply)

#### 5.24.2.2 String service_instance_name

the name of the service instance

#### 5.24.2.3 string [ ] topic_aliases

a sequence of aliases to support derived interfaces

## 5.25 RTPSReaderQosPolicy Struct Reference

### 5.25.1 Description

Allows configuration of the internal RTPS Reader behavior. Specific to the RTPS wire protocol.

### Public Attributes

- Duration_t heartbeat_response_delay
- byte accept_batch_msg
- byte send_typecode
- byte send_initial_nack
- uint precache_max_samples

### 5.25.2 Member Data Documentation

#### 5.25.2.1 byte accept_batch_msg

accept_batch_msg

#### 5.25.2.2 Duration_t heartbeat_response_delay

heartbeat_response_delay

#### 5.25.2.3 uint precache_max_samples

precache_max_samples

#### 5.25.2.4 byte send_initial_nack

send_initial_nack

#### 5.25.2.5 byte send_typecode

send_typecode

## 5.26 RTPSWriterQosPolicy Struct Reference

### 5.26.1 Description

Allows configuration of the internal RTPS Writer behavior. Specific to the RTPS wire protocol.

### Public Attributes

- Duration_t heartbeat_period
- Duration_t nack_response_delay
- Duration_t nack_suppress_delay
- Duration_t ack_deadline
- uint min_buffer_size
- uint max_buffer_size
- byte apply_filters
- byte enable_batch_msg
- byte send_typecode

### 5.26.2 Member Data Documentation

#### 5.26.2.1 Duration_t ack_deadline

ack_deadline

#### 5.26.2.2 byte apply_filters

apply_filters

#### 5.26.2.3 byte enable_batch_msg

enable_batch_msg

#### 5.26.2.4 Duration_t heartbeat_period

heartbeat period

#### 5.26.2.5 uint max_buffer_size

max_buffer_size

#### 5.26.2.6 uint min_buffer_size

min_buffer_size

#### 5.26.2.7 Duration_t nack_response_delay

nack_response_delay

#### 5.26.2.8 Duration_t nack_suppress_delay

nack_suppress_delay

#### 5.26.2.9 byte send_typecode

send_typecode

## 5.27 ThreadModelQosPolicy Struct Reference

### 5.27.1 Description

Configure the threading features of CoreDX DDS

### Public Attributes

- byte use_threads
- byte create_listener_thread

### 5.27.2 Member Data Documentation

#### 5.27.2.1 byte create_listener_thread

create a distinct thread for application Listener callbacks

#### 5.27.2.2 byte use_threads

Create and operate internal threads for DDS processing

## 5.28 TimeBasedFilterQosPolicy Struct Reference

### 5.28.1 Description

Defines a filter based on time between samples. The DataReader indicates that it wants at most one sample for each instance every minimum_separation interval.

It is inconsistent for a DataReader to have a minimum_separation larger than its DEADLINE period. By default minimum_separation=0 indicating that the DataReader should be sent all values.

### Public Attributes

- Duration_t minimum_separation

### 5.28.2 Member Data Documentation

#### 5.28.2.1 Duration_t minimum_separation

minimum separation between samples per instance

## 5.29 TopicDataQosPolicy Struct Reference

### 5.29.1 Description

Allows the application to attach arbitrary information to a Topic QoS.

The value of the TOPIC_DATA is propagated and made available to applications through the built-in topics.

The topic data is implemented as a vector of Bytes, and can be used to store any application defined data.

This QoS can be used by an application in combination with the TopicListener to implement conditional matching policies.

### Public Attributes

- byte[ ] value

### 5.29.2 Member Data Documentation

#### 5.29.2.1 byte [ ] value

topic data

## 5.30 TransportPriorityQosPolicy Struct Reference

### 5.30.1 Description

A hint to the middleware to help configure the transport priority mechanism.

### Public Attributes

- int value

### 5.30.2 Member Data Documentation

#### 5.30.2.1 int value

value

## 5.31 TypeConsistencyEnforcementQosPolicy Struct Reference

### 5.31.1 Description

This QoS policy establishes the policy for establishing consistency of the Reader and Writer data types.

The reader indicates the acceptable degree of type consistency.

### Public Attributes

- TypeConsistencyKind kind

### 5.31.2 Member Data Documentation

#### 5.31.2.1 TypeConsistencyKind kind

Kind of Type Consistency Enforcement

## 5.32 UserDataQosPolicy Struct Reference

### 5.32.1 Description

Allows the application to attach arbitrary information to a DomainParticipant, DataWriter or DataReader.

The value of the USER_DATA is propagated and made available to applications through the built-in topics.

The user data is implemented as a vector of Bytes, and can be used to store any application defined data.

This QoS can be used by an application in combination with the listeners to implement conditional matching policies.

### Public Attributes

- byte[ ] value

### 5.32.2 Member Data Documentation

#### 5.32.2.1 byte [ ] value

user data

## 5.33 WriterDataLifecycleQosPolicy Struct Reference

### 5.33.1 Description

Specifies the lifecycle behavior of data instances managed by the DataWriter. If autodispose_unregistered_instances is true, then the DataWriter will automatically dispose any instances that are unregistered. **Note:** When a DataWriter is deleted, it will automatically unregister all of its instances. With this policy == true, then all instances will also be disposed.

### Public Attributes

- bool autodispose_unregistered_instances

### 5.33.2 Member Data Documentation

#### 5.33.2.1 bool autodispose_unregistered_instances

automatically dispose instance when unregistered

## 5.34 DestinationOrderQosPolicyKind Enum Reference

### 5.34.1 Description

Valid kinds of DestinationOrder Qos.

### Public Attributes

- BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS
- BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS

### 5.34.2 Member Data Documentation

#### 5.34.2.1 BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS

ordering kind

#### 5.34.2.2 BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS

ordering kind

## 5.35 DiscoveryQosPolicyDiscoveryKind Enum Reference

### 5.35.1 Description

Valid kinds of Discovery Qos.

### Public Attributes

- PEER_DISCOVERY_QOS
- CENTRAL_DISCOVERY_QOS

### 5.35.2 Member Data Documentation

#### 5.35.2.1 CENTRAL_DISCOVERY_QOS

discovery kind

#### 5.35.2.2 PEER_DISCOVERY_QOS

discovery kind

## 5.36 DurabilityQosPolicyKind Enum Reference

### 5.36.1 Description

Valid kinds of Durability Qos.

### Public Attributes

- VOLATILE_DURABILITY_QOS
- TRANSIENT_LOCAL_DURABILITY_QOS
- TRANSIENT_DURABILITY_QOS
- PERSISTENT_DURABILITY_QOS

### 5.36.2 Member Data Documentation

#### 5.36.2.1 PERSISTENT_DURABILITY_QOS

durability kind

#### 5.36.2.2 TRANSIENT_DURABILITY_QOS

durability kind

#### 5.36.2.3 TRANSIENT_LOCAL_DURABILITY_QOS

durability kind

#### 5.36.2.4 VOLATILE_DURABILITY_QOS

durability kind

## 5.37 HistoryQosPolicyKind Enum Reference

### 5.37.1 Description

Valid kinds of History Qos.

### Public Attributes

- KEEP_LAST_HISTORY_QOS
- KEEP_ALL_HISTORY_QOS

### 5.37.2 Member Data Documentation

#### 5.37.2.1 KEEP_ALL_HISTORY_QOS

Keep all samples for each instance (up to history depth)

#### 5.37.2.2 KEEP_LAST_HISTORY_QOS

Keep most recent sample for each instance

## 5.38 LivelinessQosPolicyKind Enum Reference

### 5.38.1 Description

The 'kinds' of Liveliness Qos Policy supported.

### Public Attributes

- AUTOMATIC_LIVELINESS_QOS
- MANUAL_BY_PARTICIPANT_LIVELINESS_QOS
- MANUAL_BY_TOPIC_LIVELINESS_QOS

### 5.38.2 Member Data Documentation

#### 5.38.2.1 AUTOMATIC_LIVELINESS_QOS

liveliness kind

#### 5.38.2.2 MANUAL_BY_PARTICIPANT_LIVELINESS_QOS

liveliness kind

#### 5.38.2.3 MANUAL_BY_TOPIC_LIVELINESS_QOS

liveliness kind

## 5.39 LocatorKind Enum Reference

### 5.39.1 Description

LocatorKind is used to indicate the type of a Locator (UDPV4, TCPV4, etc)

### Public Attributes

- RESERVED = 0
- UDPV4_LOCATOR = 1
- UDPV6_LOCATOR = 2
- SHM_LOCATOR = 3
- UDS_LOCATOR = 4
- TCPV4_LOCATOR = 0x8001
- TCPV6_LOCATOR = 0x8002

### 5.39.2 Member Data Documentation

#### 5.39.2.1 RESERVED = 0

Invalid locator kind

#### 5.39.2.2 SHM_LOCATOR = 3

Shared Memory (not currently supported)

#### 5.39.2.3 TCPV4_LOCATOR = 0x8001

TCP with v4 IP Address

#### 5.39.2.4 TCPV6_LOCATOR = 0x8002

TCP with v6 IP Address

#### 5.39.2.5 UDPV4_LOCATOR = 1

UDP with v4 IP Address

#### 5.39.2.6 UDPV6_LOCATOR = 2

UDP with v6 IP Address

#### 5.39.2.7 UDS_LOCATOR = 4

UnixDomainSocket (for Central Discovery Daemon)

## 5.40 OwnershipQosPolicyKind Enum Reference

### 5.40.1 Description

Valid kinds of Ownership Qos.

### Public Attributes

- SHARED_OWNERSHIP_QOS
- EXCLUSIVE_OWNERSHIP_QOS

### 5.40.2 Member Data Documentation

#### 5.40.2.1 EXCLUSIVE_OWNERSHIP_QOS

ownership kind

#### 5.40.2.2 SHARED_OWNERSHIP_QOS

ownership kind

## 5.41 PresentationQosPolicyAccessScopeKind Enum Reference

### 5.41.1 Description

Valid kinds of Presentation Qos.

### Public Attributes

- INSTANCE_PRESENTATION_QOS
- TOPIC_PRESENTATION_QOS
- GROUP_PRESENTATION_QOS

### 5.41.2 Member Data Documentation

#### 5.41.2.1 GROUP_PRESENTATION_QOS

presentation scope

#### 5.41.2.2 INSTANCE_PRESENTATION_QOS

presentation scope

#### 5.41.2.3 TOPIC_PRESENTATION_QOS

presentation scope

## 5.42 ReliabilityQosPolicyKind Enum Reference

### 5.42.1 Description

Valid kinds of Reliability Qos.

### Public Attributes

- BEST_EFFORT_RELIABILITY_QOS
- RELIABLE_RELIABILITY_QOS

### 5.42.2 Member Data Documentation

#### 5.42.2.1 BEST_EFFORT_RELIABILITY_QOS

reliability kind

#### 5.42.2.2 RELIABLE_RELIABILITY_QOS

reliability kind

## 5.43 TypeConsistencyKind Enum Reference

### 5.43.1 Description

Valid kinds of History Qos.

### Public Attributes

- DISALLOW_TYPE_COERCION
- ALLOW_TYPE_COERCION

### 5.43.2 Member Data Documentation

#### 5.43.2.1 ALLOW_TYPE_COERCION

allow type coercion

#### 5.43.2.2 DISALLOW_TYPE_COERCION

no type coercion

# Chapter 6

# QoS Provider

The CoreDX DDS C# language mapping does not yet include an implementation of QosProvider.

# Chapter 7

# DDS Listeners

## 7.1 DataReaderListener Class Reference

### 7.1.1 Description

The DataReaderListener provides asynchronous notification of DataReader events.

This listener can be installed during DataReader creation, Subscriber.create_datareader(), as well as by calling DataReader.set_listener().

**Note**

The listener callback methods should be lightweight and should not block. If a callback method blocks, it will block all other callback operations within the same DomainParticipant.

Inherited by ServiceProxy< TReq, TRep >.ServiceProxyDRListener `[private]`.

### Public Attributes

- requested_deadline_missed_delegate on_requested_deadline_missed
- requested_incompatible_qos_delegate on_requested_incompatible_qos
- sample_rejected_delegate on_sample_rejected
- liveliness_changed_delegate on_liveliness_changed
- data_available_delegate on_data_available
- subscription_matched_delegate on_subscription_matched
- sample_lost_delegate on_sample_lost

### 7.1.2 Member Data Documentation

#### 7.1.2.1 data_available_delegate on_data_available

listener delegate

#### 7.1.2.2 liveliness_changed_delegate on_liveliness_changed

listener delegate

#### 7.1.2.3 requested_deadline_missed_delegate on_requested_deadline_missed

listener delegate

#### 7.1.2.4 requested_incompatible_qos_delegate on_requested_incompatible_qos

listener delegate

#### 7.1.2.5 sample_lost_delegate on_sample_lost

listener delegate

#### 7.1.2.6 sample_rejected_delegate on_sample_rejected

listener delegate

#### 7.1.2.7 subscription_matched_delegate on_subscription_matched

listener delegate

## 7.2 DataWriterListener Class Reference

### 7.2.1 Description

The DataWriterListener provides asynchronous notification of DataWriter events.

This listener can be installed during DataWriter creation, Publisher.create_datawriter(), as well as by calling DataWriter.set_listener().

**Note**

The listener callback methods should be lightweight and should not block. If a callback method blocks, it will block all other callback operations within the same DomainParticipant.

Inherited by ServiceProxy< TReq, TRep >.ServiceProxyDWListener `[private]`.

### Public Attributes

- offered_deadline_missed_delegate on_offered_deadline_missed
- offered_incompatible_qos_delegate on_offered_incompatible_qos
- liveliness_lost_delegate on_liveliness_lost
- publication_matched_delegate on_publication_matched

### 7.2.2 Member Data Documentation

#### 7.2.2.1 liveliness_lost_delegate on_liveliness_lost

listener delegate

#### 7.2.2.2 offered_deadline_missed_delegate on_offered_deadline_missed

listener delegate

#### 7.2.2.3 offered_incompatible_qos_delegate on_offered_incompatible_qos

listener delegate

#### 7.2.2.4 publication_matched_delegate on_publication_matched

listener delegate

## 7.3 DomainParticipantListener Class Reference

### 7.3.1 Description

The DomainParticipantListener provides asynchronous notification of DomainParticipant events.

This listener can be installed during DomainParticipant creation, DomainParticipantFactory.create_participant(), as well as by calling DomainParticipantset_listener().

**Note**

The listener callback methods should be lightweight and should not block. If a callback method blocks, it will block all other callback operations within the same DomainParticipant.

### Public Attributes

- inconsistent_topic_delegate on_inconsistent_topic
- offered_deadline_missed_delegate on_offered_deadline_missed
- offered_incompatible_qos_delegate on_offered_incompatible_qos
- liveliness_lost_delegate on_liveliness_lost
- publication_matched_delegate on_publication_matched
- requested_deadline_missed_delegate on_requested_deadline_missed
- requested_incompatible_qos_delegate on_requested_incompatible_qos
- sample_rejected_delegate on_sample_rejected
- liveliness_changed_delegate on_liveliness_changed
- data_available_delegate on_data_available
- subscription_matched_delegate on_subscription_matched
- sample_lost_delegate on_sample_lost
- data_on_readers_delegate on_data_on_readers

### 7.3.2 Member Data Documentation

#### 7.3.2.1 data_available_delegate on_data_available

listener delegate

#### 7.3.2.2 data_on_readers_delegate on_data_on_readers

listener delegate

#### 7.3.2.3 inconsistent_topic_delegate on_inconsistent_topic

listener delegate

#### 7.3.2.4 liveliness_changed_delegate on_liveliness_changed

listener delegate

#### 7.3.2.5 liveliness_lost_delegate on_liveliness_lost

listener delegate

#### 7.3.2.6 offered_deadline_missed_delegate on_offered_deadline_missed

listener delegate

#### 7.3.2.7 offered_incompatible_qos_delegate on_offered_incompatible_qos

listener delegate

#### 7.3.2.8 publication_matched_delegate on_publication_matched

listener delegate

#### 7.3.2.9 requested_deadline_missed_delegate on_requested_deadline_missed

listener delegate

#### 7.3.2.10 requested_incompatible_qos_delegate on_requested_incompatible_qos

listener delegate

#### 7.3.2.11 sample_lost_delegate on_sample_lost

listener delegate

#### 7.3.2.12 sample_rejected_delegate on_sample_rejected

listener delegate

#### 7.3.2.13 subscription_matched_delegate on_subscription_matched

listener delegate

## 7.4 PublisherListener Class Reference

### 7.4.1 Description

The PublisherListener provides asynchronous notification of Publisher events.

This listener can be installed during Publisher creation DomainParticipant.create_publisher(), as well as by calling Publisher.set_listener().

**Note**

The listener callback methods should be lightweight and should not block. If a callback method blocks, it will block all other callback operations within the same DomainParticipant.

### Public Attributes

- offered_deadline_missed_delegate on_offered_deadline_missed
- offered_incompatible_qos_delegate on_offered_incompatible_qos
- liveliness_lost_delegate on_liveliness_lost
- publication_matched_delegate on_publication_matched

### 7.4.2 Member Data Documentation

#### 7.4.2.1 liveliness_lost_delegate on_liveliness_lost

listener delegate

#### 7.4.2.2 offered_deadline_missed_delegate on_offered_deadline_missed

listener delegate

#### 7.4.2.3 offered_incompatible_qos_delegate on_offered_incompatible_qos

listener delegate

#### 7.4.2.4 publication_matched_delegate on_publication_matched

listener delegate

## 7.5 SubscriberListener Class Reference

### 7.5.1 Description

The SubscriberListener provides asynchronous notification of Subscriber events.

This listener can be installed during Subscriber creation, DomainParticipant.create_subscriber() as well as by calling Subscriber.set_listener().

**Note**

The listener callback methods should be lightweight and should not block. If a callback method blocks, it will block all other callback operations within the same DomainParticipant.

### Public Attributes

- requested_deadline_missed_delegate on_requested_deadline_missed
- requested_incompatible_qos_delegate on_requested_incompatible_qos
- sample_rejected_delegate on_sample_rejected
- liveliness_changed_delegate on_liveliness_changed
- data_available_delegate on_data_available
- subscription_matched_delegate on_subscription_matched
- sample_lost_delegate on_sample_lost
- data_on_readers_delegate on_data_on_readers

### 7.5.2 Member Data Documentation

#### 7.5.2.1 data_available_delegate on_data_available

listener delegate

#### 7.5.2.2 data_on_readers_delegate on_data_on_readers

listener delegate

#### 7.5.2.3 liveliness_changed_delegate on_liveliness_changed

listener delegate

#### 7.5.2.4 requested_deadline_missed_delegate on_requested_deadline_missed

listener delegate

#### 7.5.2.5 requested_incompatible_qos_delegate on_requested_incompatible_qos

listener delegate

#### 7.5.2.6 sample_lost_delegate on_sample_lost

listener delegate

#### 7.5.2.7 sample_rejected_delegate on_sample_rejected

listener delegate

#### 7.5.2.8 subscription_matched_delegate on_subscription_matched

listener delegate

## 7.6 TopicListener Class Reference

### 7.6.1 Description

The TopicListener provides asynchronous notification of Topic events.

This listener can be installed during Topic creation (DomainParticipant.create_topic() and related) as well as by calling Topic.set_listener().

**Note**

The listener callback methods should be lightweight and should not block. If a callback method blocks, it will block all other callback operations within the same DomainParticipant.

### Public Attributes

- inconsistent_topic_delegate on_inconsistent_topic

### 7.6.2 Member Data Documentation

#### 7.6.2.1 inconsistent_topic_delegate on_inconsistent_topic

listener delegate

# Chapter 8

# DDS Status Structures

These data types are used to communicate status information in the Listener and get_XYZ_status() methods.

## 8.1 InconsistentTopicStatus Class Reference

### 8.1.1 Description

Status related to the on_inconsistent_topic listener methods of the TopicListener structure.

#### Public Attributes

- int total_count
- int total_count_change

### 8.1.2 Member Data Documentation

#### 8.1.2.1 int total_count

Cummulative count of the discovered Topics having a matching name and inconsistent characteristics.

#### 8.1.2.2 int total_count_change

Change in **total_count** since the last time the listener was called or status was read.

## 8.2 LivelinessChangedStatus Class Reference

### 8.2.1 Description

Status related to the on_liveliness_changed listener methods of the DataReader, Subscriber, and DomainParticipant structures.

### Public Attributes

- int alive_count
- int not_alive_count
- int alive_count_change
- int not_alive_count_change
- InstanceHandle_t last_publication_handle

### 8.2.2 Member Data Documentation

#### 8.2.2.1 int alive_count

The number of 'active' DataWriters matched to this DataReader.

#### 8.2.2.2 int alive_count_change

Change in **alive_count** since the last time the listener was called or status was read.

#### 8.2.2.3 InstanceHandle_t last_publication_handle

Handle identifying the most recent DataWriter whose liveliness changed.

#### 8.2.2.4 int not_alive_count

The number of 'not-alive' DataWriters matched to this DataReader.

#### 8.2.2.5 int not_alive_count_change

Change in **not_alive_count** since the last time the listener was called or status was read.

## 8.3 LivelinessLostStatus Class Reference

### 8.3.1 Description

Status related to the on_liveliness_lost listener methods of the DataWriter, Publisher, and DomainParticipant structures.

### Public Attributes

- int total_count
- int total_count_change

### 8.3.2 Member Data Documentation

#### 8.3.2.1 int total_count

Cummulative number of times that an 'alive' DataWriter became not alive.

#### 8.3.2.2 int total_count_change

Change in **total_count** since the last time the listener was called or status was read.

## 8.4 OfferedDeadlineMissedStatus Class Reference

### 8.4.1 Description

Status related to the on_offered_deadline_missed listener methods of the DataWriter, Publisher, and DomainParticipant structures.

### Public Attributes

- int total_count
- int total_count_change
- InstanceHandle_t last_instance_handle

### 8.4.2 Member Data Documentation

#### 8.4.2.1 InstanceHandle_t last_instance_handle

Handle identifying the most recent instance whose deadline was missed.

#### 8.4.2.2 int total_count

Cummulative count of the number of deadlines missed by this DataWriter.

#### 8.4.2.3 int total_count_change

Change in **total_count** since the last time the listener was called or status was read.

## 8.5 OfferedIncompatibleQosStatus Class Reference

### 8.5.1 Description

Status related to the on_offered_incompatible_qos listener methods of the DataWriter, Publisher, and DomainParticipant structures.

### Public Attributes

- int total_count
- int total_count_change
- QosPolicyId_t last_policy_id
- QosPolicyCount[ ] policies

### 8.5.2 Member Data Documentation

#### 8.5.2.1 QosPolicyId_t last_policy_id

Id of the most recent requested incompatible QoS policy.

#### 8.5.2.2 QosPolicyCount [ ] policies

A list of QoS policies and the total number of times each QoS policy was found to be incompatible.

#### 8.5.2.3 int total_count

Cummulative count of the number of DataWriters discovered having matching Topic and incompatible QoS.

#### 8.5.2.4 int total_count_change

Change in **total_count** since the last time the listener was called or status was read.

## 8.6 PublicationMatchedStatus Class Reference

### 8.6.1 Description

Status related to the on_publication_matched listener methods of the DataWriter, Publisher, and DomainParticipant structures.

### Public Attributes

- int total_count
- int total_count_change
- int current_count
- int current_count_change

### 8.6.2 Member Data Documentation

#### 8.6.2.1 int current_count

The current number of DataReaders matched to the DataWriter.

#### 8.6.2.2 int current_count_change

Change in **current_count** since the last time the listener was called or status was read.

#### 8.6.2.3 int total_count

Cummulative count of the number of times this DataWriter has discovered a matching DataReader.

#### 8.6.2.4 int total_count_change

Change in **total_count** since the last time the listener was called or status was read.

## 8.7 RequestedDeadlineMissedStatus Class Reference

### 8.7.1 Description

Status related to the on_requested_deadline_missed listener methods of the DataReader, Subscriber, and DomainParticipant structures.

### Public Attributes

- int total_count
- int total_count_change
- InstanceHandle_t last_instance_handle

### 8.7.2 Member Data Documentation

#### 8.7.2.1 InstanceHandle_t last_instance_handle

Handle identifying the most recent instance whose deadline was missed.

#### 8.7.2.2 int total_count

Cummulative count of the number of detected deadline misses for any instance read by the DataReader.

#### 8.7.2.3 int total_count_change

Change in **total_count** since the last time the listener was called or status was read.

## 8.8 RequestedIncompatibleQosStatus Class Reference

### 8.8.1 Description

Status related to the on_requested_incompatible_qos listener methods of the DataReader, Subscriber, and DomainParticipant structures.

### Public Attributes

- int total_count
- int total_count_change
- QosPolicyId_t last_policy_id
- QosPolicyCount[ ] policies

### 8.8.2 Member Data Documentation

#### 8.8.2.1 QosPolicyId_t last_policy_id

Handle identifying the most recent QoS policy detected to be incompatible.

#### 8.8.2.2 QosPolicyCount [ ] policies

A list of QoS policies and the total number of times each QoS policy was found to be incompatible.

#### 8.8.2.3 int total_count

Cummulative count of the number of discovered DataReaders having matching Topic and incompatible QoS.

#### 8.8.2.4 int total_count_change

Change in **total_count** since the last time the listener was called or status was read.

## 8.9 SampleLostStatus Class Reference

### 8.9.1 Description

Status related to the on_sample_lost listener methods of the DataReader, Subscriber, and DomainParticipant structures.

### Public Attributes

- int total_count
- int total_count_change

### 8.9.2 Member Data Documentation

#### 8.9.2.1 int total_count

Cummulative count of all samples lost under the Topic.

#### 8.9.2.2 int total_count_change

Change in **total_count** since the last time the listener was called or status was read.

## 8.10 SampleRejectedStatus Class Reference

### 8.10.1 Description

Status related to the on_sample_rejected listener methods of the DataReader, Subscriber, and DomainParticipant structures.

### Public Attributes

- int total_count
- int total_count_change
- SampleRejectedStatusKind last_reason
- InstanceHandle_t last_instance_handle

### 8.10.2 Member Data Documentation

#### 8.10.2.1 InstanceHandle_t last_instance_handle

The handle of the instance associated with the most recently rejected sample.

#### 8.10.2.2 SampleRejectedStatusKind last_reason

The reason for rejecting the most recently rejected sample.

#### 8.10.2.3 int total_count

Cummulative count of samples rejected by the DataReader.

#### 8.10.2.4 int total_count_change

Change in **total_count** since the last time the listener was called or status was read.

## 8.11 SubscriptionMatchedStatus Class Reference

### 8.11.1 Description

Status related to the on_subscription_matched listener methods of the DataReader, Subscriber, and DomainParticipant structures.

### Public Attributes

- int total_count
- int total_count_change
- int current_count
- int current_count_change

### 8.11.2 Member Data Documentation

#### 8.11.2.1 int current_count

The current number of DataWriters matched to the DataReader.

#### 8.11.2.2 int current_count_change

Change in **current_count** since the last time the listener was called or status was read.

#### 8.11.2.3 int total_count

Cummulative count of the number of times this DataReader has discovered a matching DataWriter.

#### 8.11.2.4 int total_count_change

Change in **total_count** since the last time the listener was called or status was read.

## 8.12 SampleRejectedStatusKind Enum Reference

### 8.12.1 Description

Indicator of the reason for a sample rejection.

#### Public Attributes

- NOT_REJECTED
- REJECTED_BY_INSTANCE_LIMIT
- REJECTED_BY_SAMPLES_LIMIT
- REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT

### 8.12.2 Member Data Documentation

#### 8.12.2.1 NOT_REJECTED

rejected status

#### 8.12.2.2 REJECTED_BY_INSTANCE_LIMIT

rejected status

#### 8.12.2.3 REJECTED_BY_SAMPLES_LIMIT

rejected status

#### 8.12.2.4 REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT

rejected status

# Chapter 9

# DDS Samples

The data types documented in this section are currently used only by the RPC API.

## 9.1 LoanedSamples< T > Class Template Reference

### 9.1.1 Description

Holds a collection of sample data and related meta-data.

The LoanedSamples object holds the loaned data and meta-data returned from a DataReader::read() operation. The object manages the 'loan' by automatically calling return_loan() on the source DataReader when the object is destroyed.

Inherits IDisposablewhere T DdsType.

## 9.2 WriteSample< T > Class Template Reference

### 9.2.1 Description

Holds sample data intended for transmission.

# Chapter 10

# Builtin DDS Types

## 10.1 BuiltinTopicKey_t Class Reference

### 10.1.1 Description

Holds the 'key' for a builtin topic

Inherits DdsType.

#### Public Attributes

- int[] value

### 10.1.2 Member Data Documentation

#### 10.1.2.1 int [] value

12 bytes of key data

## 10.2 GUID_t Class Reference

### 10.2.1 Description

A complete GUID that uniquely identifies a DDS entity.

Inherits DdsType.

### Public Attributes

- byte[] value

### 10.2.2 Member Data Documentation

#### 10.2.2.1 byte [] value

the guid bytes

## 10.3 SampleIdentity_t Class Reference

### 10.3.1 Description

Uniquely identifies a sample and its source entity.

Inherits DdsType.

### Public Attributes

- com.toc.coredx.DDS.GUID_t guid

### 10.3.2 Member Data Documentation

#### 10.3.2.1 com.toc.coredx.DDS.GUID_t guid

The source of the sample. The sequence number of the sample

## 10.4 SequenceNumber_t Class Reference

### 10.4.1 Description

A sequence number that uniquely identifies a sample within the scope of a writer cache.

Inherits DdsType.

### Public Attributes

- int high
- uint low

### 10.4.2 Member Data Documentation

#### 10.4.2.1 int high

The 'high' 32 bits of the sequence number

#### 10.4.2.2 uint low

The 'low' 32 bits of the sequence number

## 10.5 QosPolicyId_t Enum Reference

### 10.5.1 Description

QosPolicyId_t is used to uniquely indicate a QoS Policy.

#### Public Attributes

- USERDATA_QOS_POLICY_ID = 1
- DURABILITY_QOS_POLICY_ID
- PRESENTATION_QOS_POLICY_ID
- DEADLINE_QOS_POLICY_ID
- LATENCYBUDGET_QOS_POLICY_ID
- OWNERSHIP_QOS_POLICY_ID
- OWNERSHIPSTRENGTH_QOS_POLICY_ID
- LIVELINESS_QOS_POLICY_ID
- TIMEBASEDFILTER_QOS_POLICY_ID
- PARTITION_QOS_POLICY_ID
- RELIABILITY_QOS_POLICY_ID
- DESTINATIONORDER_QOS_POLICY_ID
- HISTORY_QOS_POLICY_ID
- RESOURCELIMITS_QOS_POLICY_ID
- ENTITYFACTORY_QOS_POLICY_ID
- WRITERDATALIFECYCLE_QOS_POLICY_ID
- READERDATALIFECYCLE_QOS_POLICY_ID
- TOPICDATA_QOS_POLICY_ID
- GROUPDATA_QOS_POLICY_ID
- TRANSPORTPRIORITY_QOS_POLICY_ID
- LIFESPAN_QOS_POLICY_ID
- DURABILITYSERVICE_QOS_POLICY_ID
- DATA_REPRESENTATION_QOS_POLICY_ID

### 10.5.2 Member Data Documentation

#### 10.5.2.1 DATA_REPRESENTATION_QOS_POLICY_ID

data representation

#### 10.5.2.2 DEADLINE_QOS_POLICY_ID

deadline

#### 10.5.2.3 DESTINATIONORDER_QOS_POLICY_ID

destination order

#### 10.5.2.4 DURABILITY_QOS_POLICY_ID

durability

#### 10.5.2.5 DURABILITYSERVICE_QOS_POLICY_ID

durability service

#### 10.5.2.6 ENTITYFACTORY_QOS_POLICY_ID

entity factory

#### 10.5.2.7 GROUPDATA_QOS_POLICY_ID

group data

#### 10.5.2.8 HISTORY_QOS_POLICY_ID

history

#### 10.5.2.9 LATENCYBUDGET_QOS_POLICY_ID

latency budget

#### 10.5.2.10 LIFESPAN_QOS_POLICY_ID

lifespan

#### 10.5.2.11 LIVELINESS_QOS_POLICY_ID

liveliness

#### 10.5.2.12 OWNERSHIP_QOS_POLICY_ID

ownership

#### 10.5.2.13 OWNERSHIPSTRENGTH_QOS_POLICY_ID

ownership strength

#### 10.5.2.14 PARTITION_QOS_POLICY_ID

partition

#### 10.5.2.15 PRESENTATION_QOS_POLICY_ID

presentation

#### 10.5.2.16 READERDATALIFECYCLE_QOS_POLICY_ID

reader data lifecycle

#### 10.5.2.17 RELIABILITY_QOS_POLICY_ID

reliability

#### 10.5.2.18 RESOURCELIMITS_QOS_POLICY_ID

resource limits

#### 10.5.2.19 TIMEBASEDFILTER_QOS_POLICY_ID

time based filter

#### 10.5.2.20 TOPICDATA_QOS_POLICY_ID

topic data

#### 10.5.2.21 TRANSPORTPRIORITY_QOS_POLICY_ID

transport priority

#### 10.5.2.22 USERDATA_QOS_POLICY_ID = 1

user data

#### 10.5.2.23 WRITERDATALIFECYCLE_QOS_POLICY_ID

writer data lifecycle

## 10.6 ReturnCode_t Enum Reference

### 10.6.1 Description

ReturnCode_t is used to indicate success or failure of an API method.

#### Public Attributes

- RETCODE_OK
- RETCODE_ERROR
- RETCODE_UNSUPPORTED
- RETCODE_BAD_PARAMETER
- RETCODE_PRECONDITION_NOT_MET
- RETCODE_OUT_OF_RESOURCES
- RETCODE_NOT_ENABLED
- RETCODE_IMMUTABLE_POLICY
- RETCODE_INCONSISTENT_POLICY
- RETCODE_ALREADY_DELETED
- RETCODE_TIMEOUT
- RETCODE_NO_DATA

### 10.6.2 Member Data Documentation

#### 10.6.2.1 RETCODE_ALREADY_DELETED

already deleted

#### 10.6.2.2 RETCODE_BAD_PARAMETER

bad parameter

#### 10.6.2.3 RETCODE_ERROR

general error

#### 10.6.2.4 RETCODE_IMMUTABLE_POLICY

immutable policy

#### 10.6.2.5 RETCODE_INCONSISTENT_POLICY

inconsistent policy

#### 10.6.2.6 RETCODE_NO_DATA

no data

#### 10.6.2.7 RETCODE_NOT_ENABLED

not enabled

#### 10.6.2.8 RETCODE_OK

OK

#### 10.6.2.9 RETCODE_OUT_OF_RESOURCES

out of resources

#### 10.6.2.10 RETCODE_PRECONDITION_NOT_MET

precondition not met

#### 10.6.2.11 RETCODE_TIMEOUT

timeout

#### 10.6.2.12 RETCODE_UNSUPPORTED

unsupported

## 10.7 Duration_t Struct Reference

### 10.7.1 Description

Duration_t is used to indicate a duration of time.

### Public Attributes

- int sec
- uint nanosec

### 10.7.2 Member Data Documentation

#### 10.7.2.1 uint nanosec

nano seconds

#### 10.7.2.2 int sec

seconds

## 10.8 InstanceHandle_t Struct Reference

### 10.8.1 Description

InstanceHandle_t is used to specify a data instance in a Reader or Writer cache

### Public Attributes

- IntPtr value

### 10.8.2 Member Data Documentation

#### 10.8.2.1 IntPtr value

value

## 10.9 Property_t Struct Reference

### 10.9.1 Description

A name-value pair property. The 'propagate' flag indicates if this property should be transfered through discovery.

### Public Attributes

- string name
- string value
- bool propagate

### 10.9.2 Member Data Documentation

#### 10.9.2.1 string name

property name

#### 10.9.2.2 bool propagate

should property be sent with discovery information

#### 10.9.2.3 string value

property value

## 10.10 Time_t Struct Reference

### 10.10.1 Description

Time_t is used to specify an point in time.

### Public Attributes

- int sec
- uint nanosec

### 10.10.2 Member Data Documentation

#### 10.10.2.1 uint nanosec

nano seconds

#### 10.10.2.2 int sec

seconds

# Chapter 11

# DDS Discovery Types

## 11.1 ParticipantBuiltinTopicDataDataReader Class Reference

### 11.1.1 Description

The ParticipantBuiltinTopicDataDataReader is a built-in DataReader that can be used to access Participant Discovery information. The reader can be accessed through the special 'builtin' Subscriber via the DomainParticipant.get_builtin_subscriber() routine.

**See also**

ParticipantBuiltinTopicData
DomainParticipant::get_builtin_subscriber

Inherits DataReaderI< T >.

## 11.2 ParticipantBuiltinTopicData Class Reference

### 11.2.1 Description

The ParticipantBuiltinTopicData holds QoS information discovered for each known Participant in the domain.

Inherits DdsType.

### Public Attributes

- BuiltinTopicKey_t key
- UserDataQosPolicy user_data
- string entity_name

### 11.2.2 Member Data Documentation

#### 11.2.2.1 string entity_name

The entity_name associated with the participant

#### 11.2.2.2 BuiltinTopicKey_t key

The GUID (12 byte prefix) that uniquely identifies the participant

#### 11.2.2.3 UserDataQosPolicy user_data

The user_data associated with the participant

## 11.3 PublicationBuiltinTopicDataDataReader Class Reference

### 11.3.1 Description

The PublicationBuiltinTopicDataDataReader is a built-in DataReader that can be used to access Publication Discovery information. The reader can be accessed through the special 'builtin' Subscriber via the DomainParticipant.get_builtin_subscriber() routine.

**See also**

PublicationBuiltinTopicData
DomainParticipant::get_builtin_subscriber

Inherits DataReaderI< T >.

## 11.4 PublicationBuiltinTopicData Class Reference

### 11.4.1 Description

The PublicationBuiltinTopicData holds QoS information discovered for each known DataWriter in the domain.

Inherits DdsType.

### Public Attributes

- BuiltinTopicKey_t participant_key
- BuiltinTopicKey_t key
- string topic_name
- string type_name
- DurabilityQosPolicy durability
- DurabilityServiceQosPolicy durability_service
- DeadlineQosPolicy deadline
- LatencyBudgetQosPolicy latency_budget
- LivelinessQosPolicy liveliness
- ReliabilityQosPolicy reliability
- LifespanQosPolicy lifespan
- UserDataQosPolicy user_data
- OwnershipQosPolicy ownership
- OwnershipStrengthQosPolicy ownership_strength
- DestinationOrderQosPolicy destination_order
- PresentationQosPolicy presentation
- PartitionQosPolicy partition
- TopicDataQosPolicy topic_data
- GroupDataQosPolicy group_data
- string entity_name
- TypecodeQosPolicy typecode
- RpcQosPolicy rpc

### 11.4.2 Member Data Documentation

#### 11.4.2.1 DeadlineQosPolicy deadline

The deadline supported by the DataWriter.

#### 11.4.2.2 DestinationOrderQosPolicy destination_order

The destination_order mode supported by the DataWriter.

#### 11.4.2.3 DurabilityQosPolicy durability

The durability settings supported by the DataWriter.

#### 11.4.2.4 DurabilityServiceQosPolicy durability_service

The durability_service configured for the DataWriter.

#### 11.4.2.5 string entity_name

The entity_name associated with the DataWriter.

#### 11.4.2.6 GroupDataQosPolicy group_data

The group_data value associated with the parent Publisher.

#### 11.4.2.7 BuiltinTopicKey_t key

The low 4 bytes of the GUID associated with the DataWriter.

#### 11.4.2.8 LatencyBudgetQosPolicy latency_budget

The latency_budget supported by the DataWriter.

#### 11.4.2.9 LifespanQosPolicy lifespan

The lifespan associated with samples written by DataWriter.

#### 11.4.2.10 LivelinessQosPolicy liveliness

The liveliness settings supported by the DataWriter.

#### 11.4.2.11 OwnershipQosPolicy ownership

The ownership mode supported by the DataWriter.

#### 11.4.2.12 OwnershipStrengthQosPolicy ownership_strength

The ownership_strength of the DataWriter (if ownership = exclusive)

#### 11.4.2.13 BuiltinTopicKey_t participant_key

The GUID (12 bytex prefix) associated with the parent participant.

#### 11.4.2.14 PartitionQosPolicy partition

The partition[s] in which the DataWriter is publishing.

#### 11.4.2.15 PresentationQosPolicy presentation

The presentation configuration supported by the DataWriter.

#### 11.4.2.16 ReliabilityQosPolicy reliability

The reliablitiy settings supported by the DataWriter.

#### 11.4.2.17 RpcQosPolicy rpc

Used by the RPC over DDS implementation.

#### 11.4.2.18 TopicDataQosPolicy topic_data

The topic_data value associated with the topic of the DataWriter.

#### 11.4.2.19 string topic_name

The name of the topic the DataWriter is publishing.

#### 11.4.2.20 string type_name

The name of the type the DataWriter is publishing.

#### 11.4.2.21 TypecodeQosPolicy typecode

The detailed information describing the data type published by the DataWriter.

#### 11.4.2.22 UserDataQosPolicy user_data

The user_data value associated with the DataWriter.

## 11.5 SubscriptionBuiltinTopicDataDataReader Class Reference

### 11.5.1 Description

The SubscriptionBuiltinTopicDataDataReader is a built-in DataReader that can be used to access Subscription Discovery information. The reader can be accessed through the special 'builtin' Subscriber via the DomainParticipant.get_builtin_subscriber() routine.

**See also**

SubscriptionBuiltinTopicData
DomainParticipant::get_builtin_subscriber

Inherits DataReaderI< T >.

## 11.6 SubscriptionBuiltinTopicData Class Reference

### 11.6.1 Description

The SubscriptionBuiltinTopicData holds QoS information discovered for each known DataReader in the domain.

Inherits DdsType.

### Public Attributes

- BuiltinTopicKey_t participant_key
- BuiltinTopicKey_t key
- string topic_name
- string type_name
- DurabilityQosPolicy durability
- DeadlineQosPolicy deadline
- LatencyBudgetQosPolicy latency_budget
- LivelinessQosPolicy liveliness
- ReliabilityQosPolicy reliability
- OwnershipQosPolicy ownership
- DestinationOrderQosPolicy destination_order
- UserDataQosPolicy user_data
- TimeBasedFilterQosPolicy time_based_filter
- PresentationQosPolicy presentation
- PartitionQosPolicy partition
- TopicDataQosPolicy topic_data
- GroupDataQosPolicy group_data
- string entity_name
- TypecodeQosPolicy typecode
- RpcQosPolicy rpc

### 11.6.2 Member Data Documentation

#### 11.6.2.1 DeadlineQosPolicy deadline

The deadline requested by the DataReader.

#### 11.6.2.2 DestinationOrderQosPolicy destination_order

The destination_order mode requested by the DataReader.

#### 11.6.2.3 DurabilityQosPolicy durability

The durability settings requested by the DataReader.

#### 11.6.2.4 string entity_name

The entity_name associated with the DataReader.

#### 11.6.2.5 GroupDataQosPolicy group_data

The group_data value associated with the parent Subscriber.

#### 11.6.2.6 BuiltinTopicKey_t key

The low 4 bytes of the GUID associated with the DataReader.

#### 11.6.2.7 LatencyBudgetQosPolicy latency_budget

The latency_budget requested by the DataReader.

#### 11.6.2.8 LivelinessQosPolicy liveliness

The liveliness settings requested by the DataReader.

#### 11.6.2.9 OwnershipQosPolicy ownership

The ownership mode requested by the DataReader.

#### 11.6.2.10 BuiltinTopicKey_t participant_key

The GUID (12 bytex prefix) associated with the parent participant.

#### 11.6.2.11 PartitionQosPolicy partition

The partition[s] in which the DataReader is subscribing.

#### 11.6.2.12 PresentationQosPolicy presentation

The presentation configuration requested by the DataReader.

#### 11.6.2.13 ReliabilityQosPolicy reliability

The reliablitiy settings requested by the DataReader.

#### 11.6.2.14 RpcQosPolicy rpc

Used by the RPC over DDS implementation.

#### 11.6.2.15 TimeBasedFilterQosPolicy time_based_filter

The time based filter applied by the DataReader.

#### 11.6.2.16 TopicDataQosPolicy topic_data

The topic_data value associated with the topic of the DataReader.

#### 11.6.2.17 string topic_name

The name of the topic the DataReader is subscribing.

#### 11.6.2.18 string type_name

The name of the type the DataReader is subscribing.

#### 11.6.2.19 TypecodeQosPolicy typecode

The detailed information describing the data type understood by the DataReader.

#### 11.6.2.20 UserDataQosPolicy user_data

The user_data value associated with the DataReader.

# Chapter 12

# Supporting Types

## 12.1 CacheStatistics Class Reference

### 12.1.1 Description

Status related to the current state of a sample cache. Applies to DataReader and DataWriter.

#### Public Attributes

- int instance_count
- int instance_alive_count
- int instance_disposed_count
- int instance_no_writers_count
- int instance_new_count
- int sample_count
- int sample_read_count
- uint state_flags

### 12.1.2 Member Data Documentation

#### 12.1.2.1 int instance_alive_count

number of alive instances

#### 12.1.2.2 int instance_count

total number of instances

#### 12.1.2.3 int instance_disposed_count

number of disposed instances

#### 12.1.2.4 int instance_new_count

number of 'new' (view state) instances

#### 12.1.2.5 int instance_no_writers_count

number of no_writer instances

#### 12.1.2.6 int sample_count

total number of samples

#### 12.1.2.7 int sample_read_count

number of 'read' samples

#### 12.1.2.8 uint state_flags

flags indicating internal state

## 12.2 Locator Struct Reference

### 12.2.1 Description

Network address

For a list of suppored LOCATOR_KINDs, refer to include/dds/dds.h

### Public Attributes

- LocatorKind kind
- string addr

### 12.2.2 Member Data Documentation

#### 12.2.2.1 string addr

IP address, for example: "10.0.1.12"

#### 12.2.2.2 LocatorKind kind

locator kind

## 12.3 ParticipantLocator Struct Reference

### 12.3.1 Description

Describes a the location and identity of a potential peer DomainParticipant

### Public Attributes

- int participant_id
- int participant_id_max
- Locator participant_locator

### 12.3.2 Member Data Documentation

#### 12.3.2.1 int participant_id

starting/lower ID of the participant(s) at the target address

#### 12.3.2.2 int participant_id_max

maximum/upper ID of the participant(s) at the target address

#### 12.3.2.3 Locator participant_locator

the target address

## 12.4 TypecodeQosPolicy Struct Reference

### 12.4.1 Description

Contains a binary representation of a data type (meta-data).

### Public Attributes

- byte[ ] value
- byte encoding

### 12.4.2 Member Data Documentation

#### 12.4.2.1 byte encoding

encoding

#### 12.4.2.2 byte [ ] value

value

# Chapter 13

# X-Types DynamicType API

The CoreDX DDS C# language mapping does not yet include an implementation of the X-Types DynamicType system.

# Chapter 14

# CoreDX DDS Transports

## 14.1 IpTransportInterface Class Reference

### 14.1.1 Description

Defines an interface for use by the UdpTransport.

**See also**

UdpTransportConfig

### Public Attributes

- Kind kind
- byte[] addr

### 14.1.2 Member Data Documentation

#### 14.1.2.1 byte [] addr

the interface address (4 bytes for IPv4, 16 bytes for IPv6)

#### 14.1.2.2 Kind kind

the interface kinds kind of interface address

## 14.2 LmtTransportConfig Class Reference

### 14.2.1 Description

Structure that holds LMT Transport configuration items.

**See also**

DomainParticipant::add_transport(Transport) add_transport()

### Public Member Functions

- LmtTransportConfig ()
- ReturnCode_t get_default_config ()
- ReturnCode_t get_env_config ()

### Public Attributes

- int so_sndbuf
- int so_rcvbuf
- uint max_tx_size
- uint max_rx_buf_size
- uint debug_flags

### 14.2.2 Constructor & Destructor Documentation

#### 14.2.2.1 LmtTransportConfig ( ) [inline]

Constructor. Initializes all configuration items with default values.

### 14.2.3 Member Function Documentation

#### 14.2.3.1 ReturnCode_t get_default_config ( ) [inline]

Initialize the LmtTransportConfig object with default values. Currently assigned values may be overwritten by defaults.

#### 14.2.3.2 ReturnCode_t get_env_config ( ) [inline]

Query for environment variables that impact lmt transport configuration. Load the values (if any) into the LmtTransport-Config object. Currently assigned values may be overwritten by values derived from environment variables.

### 14.2.4 Member Data Documentation

#### 14.2.4.1 uint debug_flags

configure debug output of the transport

#### 14.2.4.2 uint max_rx_buf_size

limit size of RX buffer (per connection)

#### 14.2.4.3 uint max_tx_size

largest LMT packet size we transmit (default: 8192, max: 64K)

#### 14.2.4.4 int so_rcvbuf

size in bytes for socket RCVBUF

#### 14.2.4.5 int so_sndbuf

size in bytes for socket SNDBUF

## 14.3 LmtTransport Class Reference

### 14.3.1 Description

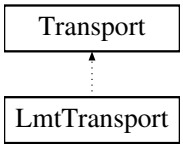
An instance of a LMT Transport that can be added to a DomainParticipant.

The LmtTransport support RTPS communication through a local machine mechanism for optimized on-platform communications.

**See also**

DomainParticipant::add_transport(Transport) add_transport()

Inheritance diagram for LmtTransport:




### Static Public Member Functions

- static LmtTransport create_transport (LmtTransportConfig transport_config)

### 14.3.2 Member Function Documentation

#### 14.3.2.1 static LmtTransport create_transport ( LmtTransportConfig *transport_config* ) [inline], [static]

Constructor

## 14.4 TcpTransportConfig Class Reference

### 14.4.1 Description

Structure that holds TCP Transport configuration items.

**See also**

DomainParticipant::add_transport(Transport) add_transport()

### Public Member Functions

- TcpTransportConfig ()
- ReturnCode_t get_default_config ()
- ReturnCode_t get_env_config ()

### Public Attributes

- short participant_index
- List< IpTransportInterface > interfaces
- bool dynamic_interfaces
- int tx_max_packet_size

### 14.4.2 Constructor & Destructor Documentation

#### 14.4.2.1 TcpTransportConfig ( ) [inline]

Constructor. Initializes all configuration items with default values.

### 14.4.3 Member Function Documentation

#### 14.4.3.1 ReturnCode_t get_default_config ( ) [inline]

Initialize the TcpTransportConfig object with default values. Currently assigned values may be overwritten by defaults.

#### 14.4.3.2 ReturnCode_t get_env_config ( ) [inline]

Query for environment variables that impact tcp transport configuration. Load the values (if any) into the TcpTransportConfig object. Currently assigned values may be overwritten by values derived from environment variables.

### 14.4.4 Member Data Documentation

#### 14.4.4.1 bool dynamic_interfaces

detect and handle changes to interface addresses

#### 14.4.4.2 List<IpTransportInterface> interfaces

default: empty -> use all available interfaces

#### 14.4.4.3 short participant_index

-1: auto detect; else force (may fail if another participant is using the ports (can't exceed 120)

#### 14.4.4.4 int tx_max_packet_size

default: 64K (to match UDP limit)

## 14.5 TcpTransport Class Reference

### 14.5.1 Description

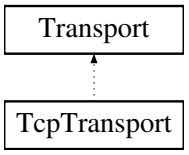
An instance of a TCP Transport that can be added to a DomainParticipant.

The TcpTransport support RTPS over a TCP/IP network. It can be used for both on-platform and off-platform communications.

**See also**

DomainParticipant::add_transport(Transport) add_transport()

Inheritance diagram for TcpTransport:




### Static Public Member Functions

- static TcpTransport create_transport (TcpTransportConfig transport_config)

### 14.5.2 Member Function Documentation

#### 14.5.2.1 static TcpTransport create_transport ( TcpTransportConfig *transport_config* ) [inline], [static]

Constructor

## 14.6 Transport Class Reference

### 14.6.1 Description

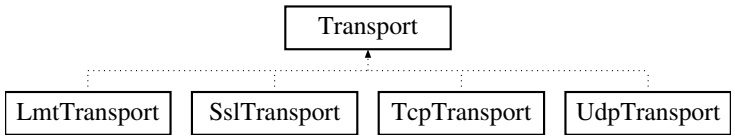
An instance of a Transport that can be added to a DomainParticipant.

The UdpTransport is the default transport used by CoreDX DDS for communications. It support RTPS over a UDP/IP network. It can be used for both on-platform and off-platform communications.

**See also**

DomainParticipant::add_transport(Transport) add_transport()

Inheritance diagram for Transport:



## 14.7 UdpTransportConfig Class Reference

### 14.7.1 Description

Structure that holds UDP Transport configuration items.

**See also**

DomainParticipant::add_transport(Transport) add_transport()

### Public Member Functions

- UdpTransportConfig ()
- ReturnCode_t get_default_config ()
- ReturnCode_t get_env_config ()

### Public Attributes

- short participant_index
- bool use_ipv4
- bool use_ipv6
- List< IpTransportInterface > interfaces
- bool dynamic_interfaces
- int rx_init_buffer_size
- int rx_max_buffer_size
- int tx_max_packet_size
- int so_rcvbuf
- int so_sndbuf
- byte[] meta_multicast_address_v4
- byte[] user_multicast_address_v4
- byte[] meta_multicast_address_v6
- byte[] user_multicast_address_v6
- byte multicast_ttl
- bool tx_meta_multicast
- bool tx_meta_unicast
- bool rx_meta_multicast
- bool rx_user_multicast
- bool advertise_meta_multicast
- bool advertise_user_multicast
- byte[] broadcast_address
- bool do_meta_broadcast
- uint debug_flags

### 14.7.2 Constructor & Destructor Documentation

#### 14.7.2.1 UdpTransportConfig ( ) [inline]

Constructor. Initializes all configuration items with default values.

### 14.7.3 Member Function Documentation

#### 14.7.3.1 ReturnCode_t get_default_config ( ) [inline]

Initialize the UdpTransportConfig object with default values. Currently assigned values may be overwritten by defaults.

#### 14.7.3.2 ReturnCode_t get_env_config ( ) [inline]

Query for environment variables that impact udp transport configuration. Load the values (if any) into the UdpTransportConfig object. Currently assigned values may be overwritten by values derived from environment variables.

### 14.7.4 Member Data Documentation

#### 14.7.4.1 bool advertise_meta_multicast

advertise we can RX META MULTICAST

#### 14.7.4.2 bool advertise_user_multicast

advertise we can RX USER MULTICAST

#### 14.7.4.3 byte [ ] broadcast_address

4 byte IPv4 address. Default: 255.255.255.255

#### 14.7.4.4 uint debug_flags

adjust the debug output from the transport

#### 14.7.4.5 bool do_meta_broadcast

enable broadcast of META (DPD discovery) data default: 0 (off)

#### 14.7.4.6 bool dynamic_interfaces

detect and handle changes to interface addresses

#### 14.7.4.7 List<IpTransportInterface> interfaces

default: empty -> use all available interfaces

#### 14.7.4.8 byte [ ] meta_multicast_address_v4

4 byte IPv4 address for meta (discovery) traffic. Default: [ 239 255 0 1 ] per the standard

#### 14.7.4.9 byte [ ] meta_multicast_address_v6

16 byte IPv6 address for meta (discovery) traffic. Default: [ ff03:0000:0000:0000:0000:efff:0001 ]

#### 14.7.4.10 byte multicast_ttl

default: 1 (0: disable all MCAST TX)

#### 14.7.4.11 short participant_index

-1: auto detect; else force (may fail if another participant is using the ports (can't exceed 120)

#### 14.7.4.12 int rx_init_buffer_size

initial size of data buffer

#### 14.7.4.13 int rx_max_buffer_size

maximum size of data buffer

#### 14.7.4.14 bool rx_meta_multicast

enable META MULTICAST (discovery) RX

#### 14.7.4.15 bool rx_user_multicast

enable USER MULTICAST (data) RX

#### 14.7.4.16 int so_rcvbuf

socket RCVBUF size (set to -1 to use OS default)

#### 14.7.4.17 int so_sndbuf

socket SNDBUF size (set to -1 to use OS default)

#### 14.7.4.18 int tx_max_packet_size

default: 64K (udp limit)

#### 14.7.4.19 bool tx_meta_multicast

enable META MULTICAST (discovery) TX

#### 14.7.4.20 bool tx_meta_unicast

enable META UNICAST (discovery) TX

#### 14.7.4.21 bool use_ipv4

Support IPv4 communications (default ON (1))

#### 14.7.4.22 bool use_ipv6

Support IPv4 communications (default OFF(0))

#### 14.7.4.23 byte [ ] user_multicast_address_v4

4 byte IPv4 address for user traffic. Default: [ 239 255 0 1 ] per the standard

#### 14.7.4.24 byte [ ] user_multicast_address_v6

16 byte IPv6 address for user traffic. Default: [ ff03:0000:0000:0000:0000:efff:0001 ]

## 14.8 UdpTransport Class Reference

### 14.8.1 Description

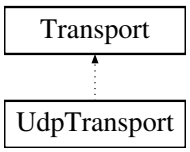
An instance of a Transport that can be added to a DomainParticipant.

The UdpTransport is the default transport used by CoreDX DDS for communications. It support RTPS over a UDP/IP network. It can be used for both on-platform and off-platform communications.

**See also**

DomainParticipant::add_transport(Transport) add_transport()

Inheritance diagram for UdpTransport:




### Static Public Member Functions

- static UdpTransport create_transport (UdpTransportConfig transport_config)

### 14.8.2 Member Function Documentation

#### 14.8.2.1 static UdpTransport create_transport ( UdpTransportConfig *transport_config* ) [inline], [static]

Constructor

# Chapter 15

# CoreDX DDS RPC over DDS API

## 15.1 ClientEndpoint< TReq, TRep > Class Template Reference

### 15.1.1 Description

A ClientEndpoint provides functions to obtain the underlying DDS entities at the client side. ClientEndpoint inherits from ServiceProxy. A ClientEndpoint shall not be instantiated directly; it can be obtained from a Client object.

Inherits ServiceProxywhere TReq RequestType< TReq, TRep >, newwhere TRep ReplyType, and new.

#### Public Member Functions

- ClientEndpoint (ClientParams cparams)
- ClientParams get_client_params ()

#### Protected Member Functions

- ReturnCode_t send_request (TReq request)
- bool receive_reply (Sample< TRep > reply, SampleIdentity_t relatedRequestId)

### 15.1.2 Constructor & Destructor Documentation

#### 15.1.2.1 ClientEndpoint ( ClientParams *cparams* ) [inline]

Default constructor

### 15.1.3 Member Function Documentation

#### 15.1.3.1 ClientParams get_client_params ( ) [inline]

Access the ClientParams defining the configuration of this ClientEndpoint.

#### 15.1.3.2 bool receive_reply ( Sample< TRep > *reply,* SampleIdentity_t *relatedRequestId* ) [inline],[protected]

Attempts to access a received reply.

This method accepts a parameter of Sample<TRep> & and a SampleIdentity_t. The call will block until a specific reply is found that matches the identity provided the 'relatedRequestId' parameter.

**Return values**

| | |
|---|---|
| *bool* | indicating if a reply was found (true) or not (false). |

#### 15.1.3.3 ReturnCode_t send_request ( TReq *request* ) [inline],[protected]

Transmits a request to any matched instances of this service This method accepts a parameter of TReq. Populates request.requestHeader

## 15.2 ClientParams Class Reference

### 15.2.1 Description

Used to pass configuration parameters when constructing a Client.

ClientParams is a valuetype that serves as a container of configuration parameters of a Client. It is designed to mimic the named-parameters feature available in some programming languages, which improves readability.

### Public Member Functions

- ClientParams ()
- ClientParams (ClientParams other)
- ClientParams service_name (String service_name)
- ClientParams instance_name (String instance_name)
- ClientParams request_topic_name (String req_topic)
- ClientParams reply_topic_name (String rep_topic)
- ClientParams datawriter_qos (DataWriterQos qos)
- ClientParams datareader_qos (DataReaderQos qos)
- ClientParams publisher (Publisher publisher)
- ClientParams subscriber (Subscriber subscriber)
- ClientParams domain_participant (DomainParticipant part)
- String service_name ()
- String instance_name ()
- String request_topic_name ()
- String reply_topic_name ()
- DataWriterQos datawriter_qos ()
- DataReaderQos datareader_qos ()
- Publisher publisher ()
- Subscriber subscriber ()
- DomainParticipant domain_participant ()

### 15.2.2 Constructor & Destructor Documentation

#### 15.2.2.1 ClientParams ( ) [inline]

Default constructor

#### 15.2.2.2 ClientParams ( ClientParams *other* ) [inline]

Copy constructor

### 15.2.3 Member Function Documentation

#### 15.2.3.1 ClientParams datareader_qos ( DataReaderQos *qos* ) [inline]

Assign the DataReaderQos configured in this instance of ClientParams.

#### 15.2.3.2 DataReaderQos datareader_qos ( ) [inline]

Access the DataReaderQos configured in this instance of ClientParams.

#### 15.2.3.3 ClientParams datawriter_qos ( DataWriterQos *qos* ) [inline]

Assign the DataWriterQos configured in this instance of ClientParams.

#### 15.2.3.4 DataWriterQos datawriter_qos ( ) [inline]

Access the DataWriterQos configured in this instance of ClientParams.

#### 15.2.3.5 ClientParams domain_participant ( DomainParticipant *part* ) [inline]

Assign the DomainParticipant configured in this instance of ClientParams.

#### 15.2.3.6 DomainParticipant domain_participant ( ) [inline]

Access the DomainParticipant configured in this instance of ClientParams.

#### 15.2.3.7 ClientParams instance_name ( String *instance_name* ) [inline]

Assign the **instance_name** configured in this instance of ClientParams.

#### 15.2.3.8 String instance_name ( ) [inline]

Access the **instance_name** configured in this instance of ClientParams.

#### 15.2.3.9 ClientParams publisher ( Publisher *publisher* ) [inline]

Assign the Publisher configured in this instance of ClientParams.

#### 15.2.3.10 Publisher publisher ( ) [inline]

Access the Publisher configured in this instance of ClientParams.

#### 15.2.3.11 ClientParams reply_topic_name ( String *rep_topic* ) [inline]

Assign the **reply_topic_name** configured in this instance of ClientParams.

#### 15.2.3.12 String reply_topic_name ( ) [inline]

Access the **reply_topic_name** configured in this instance of ClientParams.

#### 15.2.3.13 ClientParams request_topic_name ( String *req_topic* ) [inline]

Assign the **request_topic_name** configured in this instance of ClientParams.

#### 15.2.3.14 String request_topic_name ( ) [inline]

Access the **request_topic_name** configured in this instance of ClientParams.

#### 15.2.3.15 ClientParams service_name ( String *service_name* ) [inline]

Assign the **service_name** configured in this instance of ClientParams.

#### 15.2.3.16 String service_name ( ) [inline]

Access the **service_name** configured in this instance of ClientParams.

#### 15.2.3.17 ClientParams subscriber ( Subscriber *subscriber* ) [inline]

Assign the Subscriber configured in this instance of ClientParams.

#### 15.2.3.18 Subscriber subscriber ( ) [inline]

Access the Subscriber configured in this instance of ClientParams.

## 15.3 ReplierParams Class Reference

### 15.3.1 Description

Used to pass configuration parameters when constructing a Replier.

ReplierParams is a valuetype that serves as a container of configuration parameters of a Replier. It is designed to mimic the named-parameters feature available in some programming languages, which improves readability.

#### Public Member Functions

- ReplierParams ()
- ReplierParams (ReplierParams other)
- ReplierParams simple_replier_listener< TReq, TRep > (SimpleReplierListener< TReq, TRep > listener) where TReq
- ReplierParams replier_listener< TReq, TRep > (ReplierListener< TReq, TRep > listener) where TReq
- ReplierParams domain_participant (DomainParticipant participant)
- ReplierParams service_name (String service_name)
- ReplierParams instance_name (String instance_name)
- ReplierParams request_topic_name (String req_topic)
- ReplierParams reply_topic_name (String rep_topic)
- ReplierParams datawriter_qos (DataWriterQos qos)
- ReplierParams datareader_qos (DataReaderQos qos)
- ReplierParams publisher (Publisher publisher)
- ReplierParams subscriber (Subscriber subscriber)
- DomainParticipant domain_participant ()
- ListenerBase simple_replier_listener ()
- ListenerBase replier_listener ()
- String service_name ()
- String instance_name ()
- String request_topic_name ()
- String reply_topic_name ()
- DataWriterQos datawriter_qos ()
- DataReaderQos datareader_qos ()
- Publisher publisher ()
- Subscriber subscriber ()

### 15.3.2 Constructor & Destructor Documentation

#### 15.3.2.1 ReplierParams ( ) [inline]

Default constructor

#### 15.3.2.2 ReplierParams ( ReplierParams *other* ) [inline]

Copy constructor

### 15.3.3 Member Function Documentation

#### 15.3.3.1 ReplierParams datareader_qos ( DataReaderQos *qos* ) [inline]

Assign the DataReaderQos configured in this instance of ReplierParams.

#### 15.3.3.2 DataReaderQos datareader_qos ( ) [inline]

Access the DataReaderQos configured in this instance of ReplierParams.

#### 15.3.3.3 ReplierParams datawriter_qos ( DataWriterQos *qos* ) [inline]

Assign the DataWriterQos configured in this instance of ReplierParams.

#### 15.3.3.4 DataWriterQos datawriter_qos ( ) [inline]

Access the DataWriterQos configured in this instance of ReplierParams.

#### 15.3.3.5 ReplierParams domain_participant ( DomainParticipant *participant* ) [inline]

Assign the DomainParticipant configured in this instance of ReplierParams.

#### 15.3.3.6 DomainParticipant domain_participant ( ) [inline]

Access the DomainParticipant configured in this instance of ReplierParams.

#### 15.3.3.7 ReplierParams instance_name ( String *instance_name* ) [inline]

Assign the **instance_name** configured in this instance of ReplierParams.

#### 15.3.3.8 String instance_name ( ) [inline]

Access the instance_name configured in this instance of ReplierParams.

#### 15.3.3.9 ReplierParams publisher ( Publisher *publisher* ) [inline]

Assign the Publisher configured in this instance of ReplierParams.

#### 15.3.3.10 Publisher publisher ( ) [inline]

Access the Publisher configured in this instance of ReplierParams.

#### 15.3.3.11 ListenerBase replier_listener ( ) [inline]

Access the ReplierListener configured in this instance of ReplierParams.

#### 15.3.3.12 ReplierParams replier_listener< TReq, TRep > ( ReplierListener< TReq, TRep > *listener* ) [inline]

Assign the ReplierListener configured in this instance of ReplierParams.

#### 15.3.3.13 ReplierParams reply_topic_name ( String *rep_topic* ) [inline]

Assign the **reply_topic_name** configured in this instance of ReplierParams.

#### 15.3.3.14 String reply_topic_name ( ) [inline]

Access the reply_topic_name configured in this instance of ReplierParams.

#### 15.3.3.15 ReplierParams request_topic_name ( String *req_topic* ) [inline]

Assign the **request_topic_name** configured in this instance of ReplierParams.

#### 15.3.3.16 String request_topic_name ( ) [inline]

Access the request_topic_name configured in this instance of ReplierParams.

#### 15.3.3.17 ReplierParams service_name ( String *service_name* ) [inline]

Assign the **service_name** configured in this instance of ReplierParams.

#### 15.3.3.18 String service_name ( ) [inline]

Access the service_name configured in this instance of ReplierParams.

#### 15.3.3.19 ListenerBase simple_replier_listener ( ) [inline]

Access the SimpleReplierListener configured in this instance of ReplierParams.i

#### 15.3.3.20 ReplierParams simple_replier_listener< TReq, TRep > ( SimpleReplierListener< TReq, TRep > *listener* ) [inline]

Assign the SimpleReplierListener configured in this instance of ReplierParams.

#### 15.3.3.21 ReplierParams subscriber ( Subscriber *subscriber* ) [inline]

Assign the Subscriber configured in this instance of ReplierParams.

#### 15.3.3.22 Subscriber subscriber ( ) [inline]

Access the Subscriber configured in this instance of ReplierParams.

## 15.4 Replier< TReq, TRep > Class Template Reference

### 15.4.1 Description

A replier receives requests and send replies.

An instance of Replier is configured at the time of construction using ReplierParams, which is a container of configuration parameters such as domain participant, QoS, listeners and more.

Replier allows listener-based and polling-based reception of requests.

SimpleReplierListener and ReplierListener interfaces enable call-back based notification when a request is available. On the other hand, Replier provides functions to allow polling reception of requests.

Inherits ReplierBasewhere TReq RequestType, newwhere TRep ReplyType, and new.

### Public Member Functions

- Replier (ReplierParams r_params)
- void close ()
- bool is_null ()
- void send_reply (TRep reply, SampleIdentity_t related_request_id)
- bool receive_request (Sample< TReq > request, Duration_t max_wait)
- LoanedSamples< TReq > receive_requests (Duration_t max_wait)
- LoanedSamples< TReq > receive_requests (int min_request_count, int max_request_count, Duration_t max_wait)
- bool wait_for_requests (Duration_t max_wait)
- bool wait_for_requests (int min_count, Duration_t max_wait)
- bool take_request (Sample< TReq > request)
- LoanedSamples< TReq > take_requests (int max_samples)
- bool read_request (Sample< TReq > request)
- LoanedSamples< TReq > read_requests (int max_samples)
- ReplierParams get_replier_params ()
- bool receive_nondata_samples (bool enable)
- DataReaderI< TReq > get_request_datareader ()
- DataWriterI< TRep > get_reply_datawriter ()

### 15.4.2 Constructor & Destructor Documentation

#### 15.4.2.1 Replier ( ReplierParams *r_params* ) [inline]

Constructor that accepts ReplierParams configuration.

### 15.4.3 Member Function Documentation

#### 15.4.3.1 void close ( ) [inline]

Release the underlying resources including any DDS entities.

The RPCEntity will no longer be useful and will not participate in any communication.

#### 15.4.3.2 ReplierParams get_replier_params ( ) [inline]

Access the ReplierParams values that define the configuration of the Replier.

#### 15.4.3.3 DataWriterI<TRep> get_reply_datawriter ( ) [inline]

Access the underlying DataWriter used to send Replies.

#### 15.4.3.4 DataReaderI<TReq> get_request_datareader ( ) [inline]

Access the underlying DataReader used to receive Requests.

#### 15.4.3.5 bool is_null ( ) [inline]

Indicates if the entity has been 'closed'.

If is_null() is true, then the underlying dds entities have been released and the RPCEntity is no longer useful for operations.

#### 15.4.3.6 bool read_request ( Sample< TReq > *request* ) [inline]

Attempt to access the next request.

This may will block until a request is available.

**Return values**

| | |
|---|---|
| *true* | if able to access a request. |
| *false* | otherwise |

#### 15.4.3.7 LoanedSamples<TReq> read_requests ( int *max_samples* ) [inline]

Attempt to access up to 'max_samples' requests.

This call may block if no requests are available.

**Return values**

| | |
|---|---|
| *collection* | of requests |

#### 15.4.3.8 bool receive_nondata_samples ( bool *enable* ) [inline]

Toggle the underlying entities to deliver non-data samples to the application

**Not Yet Supported**

#### 15.4.3.9 bool receive_request ( Sample< TReq > *request,* Duration_t *max_wait* ) [inline]

Attempt to access the next request.

This may block for up to 'max_wait' if no request is immediately available.

#### 15.4.3.10 LoanedSamples<TReq> receive_requests ( Duration_t *max_wait* ) [inline]

Attempt to access any available requests.

This may block for up to 'max_wait' if no requests are immediately available.

#### 15.4.3.11 LoanedSamples<TReq> receive_requests ( int *min_request_count,* int *max_request_count,* Duration_t *max_wait* ) [inline]

Attempt to access up to 'max_request_count' requests.

This may block for up to 'max_wait' if there are not at least 'min_request_count' requests immediately available.

#### 15.4.3.12 void send_reply ( TRep *reply,* SampleIdentity_t *related_request_id* ) [inline]

Sends the provided 'reply' associated with 'related_request_id'.

#### 15.4.3.13 bool take_request ( Sample< TReq > *request* ) [inline]

Attempt to access the next request.

This call will not block.

**Return values**

| | |
|---|---|
| *true* | if able to access a request. |
| *false* | otherwise |

#### 15.4.3.14 LoanedSamples<TReq> take_requests ( int *max_samples* ) [inline]

Attempt to access up to 'max_samples' requests.

This call will not block.

**Return values**

| | |
|---|---|
| *collection* | of requests (possibly empty) |

#### 15.4.3.15 bool wait_for_requests ( Duration_t *max_wait* ) [inline]

Block until at least one request is available.

This may block for up to 'max_wait'.

#### 15.4.3.16 bool wait_for_requests ( int *min_count,* Duration_t *max_wait* ) [inline]

Block until at least 'min_count requests available.

This may block for up to 'max_wait'.

## 15.5 RequesterParams Class Reference

### 15.5.1 Description

Used to pass configuration parameters when constructing a Requester.

RequesterParams is a valuetype that serves as a container of configuration parameters of a Requester. It is designed to mimic the named-parameters feature available in some programming languages, which improves readability.

### Public Member Functions

- RequesterParams ()
- RequesterParams (RequesterParams other)
- RequesterParams simple_requester_listener< TRep > (SimpleRequesterListener< TRep > listener) where TRep
- RequesterParams requester_listener< TReq, TRep > (RequesterListener< TReq, TRep > listener) where TReq
- RequesterParams domain_participant (DomainParticipant participant)
- RequesterParams publisher (Publisher publisher)
- RequesterParams subscriber (Subscriber subscriber)
- RequesterParams datawriter_qos (DataWriterQos qos)
- RequesterParams datareader_qos (DataReaderQos qos)
- RequesterParams service_name (String name)
- RequesterParams request_topic_name (String name)
- RequesterParams reply_topic_name (String name)
- DomainParticipant domain_participant ()
- Publisher publisher ()
- Subscriber subscriber ()
- DataWriterQos datawriter_qos ()
- DataReaderQos datareader_qos ()
- ListenerBase simple_requester_listener ()
- ListenerBase requester_listener ()
- String service_name ()
- String request_topic_name ()
- String reply_topic_name ()

### 15.5.2 Constructor & Destructor Documentation

#### 15.5.2.1 RequesterParams ( ) [inline]

Default constructor

#### 15.5.2.2 RequesterParams ( RequesterParams *other* ) [inline]

Copy constructor

### 15.5.3 Member Function Documentation

#### 15.5.3.1 RequesterParams datareader_qos ( DataReaderQos *qos* ) [inline]

Assign the DataReaderQos to use.

This QoS will be used when the Requester creates a DataReader.

#### 15.5.3.2 DataReaderQos datareader_qos ( ) [inline]

Access the DataReaderQos configured in this instance of RequesterParams.

#### 15.5.3.3 RequesterParams datawriter_qos ( DataWriterQos *qos* ) [inline]

Assign the DataWriterQos to use.

This QoS will be used when the Requester creates a DataWriter.

#### 15.5.3.4 DataWriterQos datawriter_qos ( ) [inline]

Access the DataWriterQos configured in this instance of RequesterParams.

#### 15.5.3.5 RequesterParams domain_participant ( DomainParticipant *participant* ) [inline]

Assign the DomainParticipant to use.

The requester will use this DomainParticipant to create any required Publisher, Subscriber, DataWriter and DataReader entities.

#### 15.5.3.6 DomainParticipant domain_participant ( ) [inline]

Access the DomainParticipant configured in this instance of RequesterParams.

#### 15.5.3.7 RequesterParams publisher ( Publisher *publisher* ) [inline]

Assign the Publisher to use.

The Requester will use this publisher to create any required DataWriter entities. If not provided the Requester will create its own Publisher.

#### 15.5.3.8 Publisher publisher ( ) [inline]

Access the Publisher configured in this instance of RequesterParams.

#### 15.5.3.9 RequesterParams reply_topic_name ( String *name* ) [inline]

Assign the 'reply_topic_name'.

This overrides the default topic name generation.

#### 15.5.3.10 String reply_topic_name ( ) [inline]

Access the reply_topic_name configured in this instance of RequesterParams.

#### 15.5.3.11 RequesterParams request_topic_name ( String *name* ) [inline]

Assign the 'request_topic_name' to use as the Request topic name.

This overrides the default topic name generation.

#### 15.5.3.12 String request_topic_name ( ) [inline]

Access the request_topic_name configured in this instance of RequesterParams.

#### 15.5.3.13 ListenerBase requester_listener ( ) [inline]

Access the RequesterListener configured in this instance of RequesterParams.

#### 15.5.3.14 RequesterParams requester_listener< TReq, TRep > ( RequesterListener< TReq, TRep > *listener* ) [inline]

Assign the 'requester' listener that will be installed in the Requester.

Only one listener may be installed, either the SimpleListener or RequesterListener.

#### 15.5.3.15 RequesterParams service_name ( String *name* ) [inline]

Assign the service_name to use when creating the Request and Reply topic names.

The service_name is used when constructing the topic name used for the Request and Reply topics. If not explicitly defined by the request_topic_name() configuration item, the request topic is composed of <service_name>_Request. Similarly, if not overridden by reply_topic_name(), the reply topic name is composed of <service_name>_Reply. If the service_name() item is left unspecified, then the string "Service" is used.

#### 15.5.3.16 String service_name ( ) [inline]

Access the service_name configured in this instance of RequesterParams.

#### 15.5.3.17 ListenerBase simple_requester_listener ( ) [inline]

Access the SimpleRequesterListener configured in this instance of RequesterParams.

#### 15.5.3.18 RequesterParams simple_requester_listener< TRep > ( SimpleRequesterListener< TRep > *listener* ) [inline]

Assign the 'simple' listener that will be installed in the Requester.

Only one listener may be installed, either the SimpleListener or RequesterListener.

#### 15.5.3.19 RequesterParams subscriber ( Subscriber *subscriber* ) [inline]

Assign the Subscriber to use.

The Requester will use this subscriber to create any requried DataReader entities. If not provided, the Requester will create its own Subscriber.

#### 15.5.3.20 Subscriber subscriber ( ) [inline]

Access the Subscriber configured in this instance of RequesterParams.

## 15.6 Requester< TReq, TRep > Class Template Reference

### 15.6.1 Description

A Requester sends requests and receives replies.

An instance of a Requester is configured at the time of construction using RequesterParams, which is a container of configuration parameters, such as domain participant, QoS, listeners and more.

Requester is inherently asynchronous as sending a request and receiving its corresponding reply (or replies) are separated. Requester allows listener-based, polling-based, and future-based reception of replies.

SimpleRequesterListener<TRep> and RequesterListener<TReq, TRep> interfaces enable callback-based notification when a reply is available. On the other hand, Requester provides functions to allow polling reception of replies.

Future-based notification of replies is analogous to callback-based notification, however, no request-reply correlation is necessary because every future represents a reply to a unique request.

A requester reference may be bound to a specific service instance. Requests sent through a bound requester reference shall be sent to the bound service instance only.

Inherits ServiceProxywhere TReq RequestType< TReq, TRep >, newwhere TRep ReplyType, and new.

### Public Member Functions

- Requester ()
- Requester (RequesterParams req_params)
- ReturnCode_t send_request (TReq request)
- ReturnCode_t send_request_oneway (TReq request)
- bool receive_reply (Sample< TRep > reply, Duration_t timeout)
- bool receive_reply (Sample< TRep > reply, SampleIdentity_t relatedRequestId)
- LoanedSamples< TRep > receive_replies (Duration_t max_wait)
- LoanedSamples< TRep > receive_replies (int min_count, int max_count, Duration_t max_wait)
- bool wait_for_replies (int min_count, Duration_t max_wait)
- bool wait_for_replies (Duration_t max_wait)
- bool wait_for_replies (int min_count, Duration_t max_wait, SampleIdentity_t related_request_id)
- bool take_reply (Sample< TRep > reply)
- bool take_reply (Sample< TRep > reply, SampleIdentity_t related_request_id)
- LoanedSamples< TRep > take_replies (int max_count)
- LoanedSamples< TRep > take_replies (int max_count, SampleIdentity_t related_request_id)
- LoanedSamples< TRep > take_replies (SampleIdentity_t related_request_id)
- bool read_reply (Sample< TRep > reply)
- bool read_reply (Sample< TRep > reply, SampleIdentity_t related_request_id)
- LoanedSamples< TRep > read_replies (int max_count)
- LoanedSamples< TRep > read_replies (int max_count, SampleIdentity_t related_request_id)
- LoanedSamples< TRep > read_replies (SampleIdentity_t related_request_id)
- bool receive_nondata_samples (bool enable)
- RequesterParams get_requester_params ()
- DataWriterI< TReq > get_request_datawriter ()
- DataReaderI< TRep > get_reply_datareader ()

### 15.6.2 Constructor & Destructor Documentation

#### 15.6.2.1 Requester ( ) [inline]

Default constructor.

Without any configuration, the Requester will construct DDS Entities as required in domain '0'.

#### 15.6.2.2 Requester ( RequesterParams *req_params* ) [inline]

Constructor that accepts RequesterParams for configuration.

### 15.6.3 Member Function Documentation

#### 15.6.3.1 DataReaderI<TRep> get_reply_datareader ( ) [inline]

Obtain access to the underlying DataReader used for receiving Replies.

#### 15.6.3.2 DataWriterI<TReq> get_request_datawriter ( ) [inline]

Obtain access to the underlying DataWriter used for sending Requests.

#### 15.6.3.3 RequesterParams get_requester_params ( ) [inline]

Obtain the parameters used to configure this Reqeuster.

#### 15.6.3.4 LoanedSamples<TRep> read_replies ( int *max_count* ) [inline]

Attempts to access up to 'max_count' replies.

The available replies (up to 'max_count') are returned in a LoanedSamples<TRep> object. This routine will not block.

#### 15.6.3.5 LoanedSamples<TRep> read_replies ( int *max_count,* SampleIdentity_t *related_request_id* ) [inline]

Attempts to access up to 'max_count' replies that match 'related_request_id'.

The available matching replies (up to 'max_count') are returned in a LoanedSamples<TRep> object. This routine will not block.

#### 15.6.3.6 LoanedSamples<TRep> read_replies ( SampleIdentity_t *related_request_id* ) [inline]

Attempts to access all available replies that match 'related_request_id'.

The available matching replies are returned in a LoanedSamples<TRep> object. This routine will not block.

#### 15.6.3.7 bool read_reply ( Sample< TRep > *reply* ) [inline]

Attempts to access the next available reply.

This routine will block until a reply is available. This routines copies the reply data into the provided Sample<TRep> 'reply'.

#### 15.6.3.8 bool read_reply ( Sample< TRep > *reply,* SampleIdentity_t *related_request_id* ) [inline]

Attempts to access the next available reply that matches 'related_reqeust_id'.

This routine will block until a matching reply is available.

#### 15.6.3.9 bool receive_nondata_samples ( bool *enable* ) [inline]

Toggle whether non-data samples are presented to the application.

**Note**

This is of little utility, as the only non-data samples are unregister, dispose, and not-alive indications which are uncommon with an un-keyed datatype.

#### 15.6.3.10 LoanedSamples<TRep> receive_replies ( Duration_t *max_wait* ) [inline]

Attempts to access multiple received replies.

This method will access all available replies and return them in a LoanedSamples<TRep> object. If no replies are currently available, the routine will block for up to 'max_wait'.

**Return values**

| | |
|---|---|
| *bool* | indicating if replies were found (true) or not (false). |

#### 15.6.3.11 LoanedSamples<TRep> receive_replies ( int *min_count,* int *max_count,* Duration_t *max_wait* ) [inline]

Attempts to access at least 'min_count' replies.

This method will access all available replies up to 'max_count' and return them in a LoanedSamples<TRep> object. If at least 'min_count' replies are currently not available, the routine will block for up to 'max_wait'.

**Return values**

| | |
|---|---|
| *bool* | indicating if replies were found (true) or not (false). |

#### 15.6.3.12 bool receive_reply ( Sample< TRep > *reply,* Duration_t *timeout* ) [inline]

Attempts to access a received reply.

This method accepts a parameter of Sample<TRep> & and a 'timeout'. The call will block until a reply is found, or the timeout period elapses.

**Return values**

| | |
|---|---|
| *bool* | indicating if a reply was found (true) or not (false). |

#### 15.6.3.13 bool receive_reply ( Sample< TRep > *reply,* SampleIdentity_t *relatedRequestId* ) [inline]

Attempts to access a received reply.

This method accepts a parameter of Sample<TRep> & and a SampleIdentity_t. The call will block until a specific reply is found that matches the identity provided the 'relatedRequestId' parameter.

**Return values**

| | |
|---|---|
| *bool* | indicating if a reply was found (true) or not (false). |

#### 15.6.3.14 ReturnCode_t send_request ( TReq *request* ) [inline]

Transmits a request to any matched instances of this service This method accepts a parameter of TReq. Populates request.requestHeader

#### 15.6.3.15 ReturnCode_t send_request_oneway ( TReq *request* ) [inline]

Transmits a request to any matched instances of this service This method accepts a parameter of TReq.

#### 15.6.3.16 LoanedSamples<TRep> take_replies ( int *max_count* ) [inline]

Attempts to access up to 'max_count' replies.

This method will access all available replies up to 'max_count' and return them in a LoanedSamples<TRep> object. This routine will not block.

#### 15.6.3.17 LoanedSamples<TRep> take_replies ( int *max_count,* SampleIdentity_t *related_request_id* ) [inline]

Attempts to access up to 'max_count' replies that match the 'related_request_id'.

This method will access all available replies up to 'max_count' and return them in a LoanedSamples<TRep> object. This routine will not block.

#### 15.6.3.18 LoanedSamples<TRep> take_replies ( SampleIdentity_t *related_request_id* ) [inline]

Attempts to access any replies that match the 'related_request_id'.

This method will access all available matching replies and return them in a LoanedSamples<TRep> object. This routine will not block.

#### 15.6.3.19 bool take_reply ( Sample< TRep > *reply* ) [inline]

Attempts to access a received reply.

This method accepts a parameter of Sample<TRep> & . The call will not block.

**Return values**

| | |
|---|---|
| *bool* | indicating if a reply was found (true) or not (false). |

#### 15.6.3.20 bool take_reply ( Sample< TRep > *reply,* SampleIdentity_t *related_request_id* ) [inline]

Attempts to access a received reply.

This method accepts a parameter of Sample<TRep> & . The call will not block.

**Return values**

| | |
|---|---|
| *bool* | indicating if a reply was found (true) or not (false). |

#### 15.6.3.21 bool wait_for_replies ( int *min_count,* Duration_t *max_wait* ) [inline]

Blocks until at least 'min_count' replies are available.

If at least 'min_count' replies are currently not available, the routine will block for up to 'max_wait'.

**Return values**

| | |
|---|---|
| *bool* | indicating if replies were found (true) or not (false). |

#### 15.6.3.22 bool wait_for_replies ( Duration_t *max_wait* ) [inline]

Blocks until a reply is available.

If no replies are currently available, the routine will block for up to 'max_wait'.

**Return values**

| | |
|---|---|
| *bool* | indicating if replies were found (true) or not (false). |

#### 15.6.3.23 bool wait_for_replies ( int *min_count,* Duration_t *max_wait,* SampleIdentity_t *related_request_id* ) [inline]

Blocks until at least 'min_count' replies to 'related_request_id' are available.

If at least 'min_count' matching replies are currently not available, the routine will block for up to 'max_wait'.

**Return values**

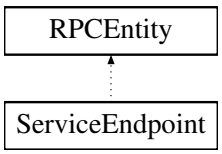
| | |
|---|---|
| *bool* | indicating if replies were found (true) or not (false). |

**Not Yet Supported** This routine currently ignores 'related_request_id', and will return with success if *any* 'min_count' replies are available.

## 15.7 ServiceEndpoint Class Reference

### 15.7.1 Description

A ServiceEndpoint provides functions to manage a service (e.g., pause, resume, close, etc.) A ServiceEndpoint shall not be instantiated directly; it can be obtained from a Service object. ServiceEndpoint is a reference type. All 'Service' implementations derive from ServiceEndpoint. Inheritance diagram for ServiceEndpoint:




### Public Member Functions

- ServiceEndpoint ()
- abstract void close ()
- abstract bool is_null ()
- abstract void pause ()
- abstract void resume ()
- ServiceStatus status ()
- ServiceParams get_service_params ()

### 15.7.2 Constructor & Destructor Documentation

#### 15.7.2.1 ServiceEndpoint ( ) [inline]

Default constructor

### 15.7.3 Member Function Documentation

#### 15.7.3.1 abstract void close ( ) [abstract]

Release the underlying resources including any DDS entities.

The RPCEntity will no longer be useful and will not participate in any communication.

Implements RPCEntity.

#### 15.7.3.2 ServiceParams get_service_params ( ) [inline]

Access the ServiceParams defining the configuration of this ServiceEndpoint.

#### 15.7.3.3 abstract bool is_null ( ) [abstract]

Indicates if the entity has been 'closed'.

If is_null() is true, then the underlying dds entities have been released and the RPCEntity is no longer useful for operations.

Implements RPCEntity.

#### 15.7.3.4 abstract void pause ( ) [abstract]

Pause the operation of the ServiceEndpoint.

This causes the ServiceEndpoint to not invoke an attached listener.

#### 15.7.3.5 abstract void resume ( ) [abstract]

Resume the operation of the ServiceEndpoint.

This causes the ServiceEndpoint to invoke any attached listeners upon reception of a request. In order to handle any requests received while the ServiceEndpoint was paused, the listener will be invoked as part of the resume() operation.

#### 15.7.3.6 ServiceStatus status ( ) [inline]

Access the current status of the ServiceEndpoint.

## 15.8 ServiceParams Class Reference

### 15.8.1 Description

Used to pass configuration parameters when constructing a Service.

ServiceParams is a valuetype that serves as a container of configuration parameters of a Service. It is designed to mimic the named-parameters feature available in some programming languages, which improves readability.

### Public Member Functions

- ServiceParams ()
- ServiceParams (ServiceParams other)
- ServiceParams service_name (string service_name)
- ServiceParams instance_name (string instance_name)
- ServiceParams request_topic_name (string req_topic)
- ServiceParams reply_topic_name (String rep_topic)
- ServiceParams datawriter_qos (DataWriterQos qos)
- ServiceParams datareader_qos (DataReaderQos qos)
- ServiceParams publisher (Publisher publisher)
- ServiceParams subscriber (Subscriber subscriber)
- ServiceParams domain_participant (DomainParticipant part)
- String service_name ()
- String instance_name ()
- String request_topic_name ()
- String reply_topic_name ()
- DataWriterQos datawriter_qos ()
- DataReaderQos datareader_qos ()
- Publisher publisher ()
- Subscriber subscriber ()
- DomainParticipant domain_participant ()

### 15.8.2 Constructor & Destructor Documentation

#### 15.8.2.1 ServiceParams ( ) [inline]

Default constructor

#### 15.8.2.2 ServiceParams ( ServiceParams *other* ) [inline]

Copy constructor

### 15.8.3 Member Function Documentation

#### 15.8.3.1 ServiceParams datareader_qos ( DataReaderQos *qos* ) [inline]

Assign the DataReaderQos configured in this instance of ServiceParams.

#### 15.8.3.2 DataReaderQos datareader_qos ( ) [inline]

Access the DataReaderQos configured in this instance of ServiceParams.

#### 15.8.3.3 ServiceParams datawriter_qos ( DataWriterQos *qos* ) [inline]

Assign the DataWriterQos configured in this instance of ServiceParams.

#### 15.8.3.4 DataWriterQos datawriter_qos ( ) [inline]

Access the DataWriterQos configured in this instance of ServiceParams.

#### 15.8.3.5 ServiceParams domain_participant ( DomainParticipant *part* ) [inline]

Assign the DomainParticipant configured in this instance of ServiceParams.

#### 15.8.3.6 DomainParticipant domain_participant ( ) [inline]

Access the DomainParticipant configured in this instance of ServiceParams.

#### 15.8.3.7 ServiceParams instance_name ( string *instance_name* ) [inline]

Assign the **instance_name** configured in this instance of ServiceParams.

#### 15.8.3.8 String instance_name ( ) [inline]

Access the **instance_name** configured in this instance of ServiceParams.

#### 15.8.3.9 ServiceParams publisher ( Publisher *publisher* ) [inline]

Assign the Publisher configured in this instance of ServiceParams.

#### 15.8.3.10 Publisher publisher ( ) [inline]

Access the Publisher configured in this instance of ServiceParams.

#### 15.8.3.11 ServiceParams reply_topic_name ( String *rep_topic* ) [inline]

Assign the **reply_topic_name** configured in this instance of ServiceParams.

#### 15.8.3.12 String reply_topic_name ( ) [inline]

Access the **reply_topic_name** configured in this instance of ServiceParams.

#### 15.8.3.13 ServiceParams request_topic_name ( string *req_topic* ) [inline]

Assign the **request_topic_name** configured in this instance of ServiceParams.

#### 15.8.3.14 String request_topic_name ( ) [inline]

Access the **request_topic_name** configured in this instance of ServiceParams.

#### 15.8.3.15 ServiceParams service_name ( string *service_name* ) [inline]

Assign the **service_name** configured in this instance of ServiceParams.

#### 15.8.3.16 String service_name ( ) [inline]

Access the **service_name** configured in this instance of ServiceParams.

#### 15.8.3.17 ServiceParams subscriber ( Subscriber *subscriber* ) [inline]

Assign the Subscriber configured in this instance of ServiceParams.

#### 15.8.3.18 Subscriber subscriber ( ) [inline]

Access the Subscriber configured in this instance of ServiceParams.

## 15.9 ServiceProxy< TReq, TRep > Class Template Reference

### 15.9.1 Description

ServiceProxy

ServiceProxy class defines type-independent operations for Requester and the Client. For example, binding to a specific instance, waiting for an instance to discover, closing the service, and more. ServiceProxy shall not be instantiated directly.

Inherits RPCEntitywhere TReq RequestType, newwhere TRep ReplyType, and new.

### Public Member Functions

- void close ()
- bool is_null ()
- void bind (String i_name)
- void unbind ()
- bool is_bound ()
- String get_bound_instance_name ()
- List< String > get_discovered_service_instances ()
- ReturnCode_t wait_for_service ()
- ReturnCode_t wait_for_service (Duration_t maxWait)
- ReturnCode_t wait_for_service (String instanceName)
- ReturnCode_t wait_for_service (Duration_t max_wait, String instanceName)
- ReturnCode_t wait_for_services (uint count)
- ReturnCode_t wait_for_services (Duration_t max_wait, uint count)
- ReturnCode_t wait_for_services (List< String > instanceNames)
- ReturnCode_t wait_for_services (Duration_t max_wait, List< String > instanceNames)

### 15.9.2 Member Function Documentation

#### 15.9.2.1 void bind ( String *i_name* ) [inline]

Bind the ServiceProxy to service(s) with the provided 'instance_name'.

The ServiceProxy will communicate only with services that have a matching 'instance_name'.

#### 15.9.2.2 void close ( ) [inline]

Release the underlying resources including any DDS entities.

The RPCEntity will no longer be useful and will not participate in any communication.

#### 15.9.2.3 String get_bound_instance_name ( ) [inline]

Provides the instance_name to which the ServiceProxy is currently bound.

#### 15.9.2.4 List<String> get_discovered_service_instances ( ) [inline]

Access a list of instance_names that have been discovered by the ServiceProxy.

The returned list is a snapshot in time of the currently known services.

#### 15.9.2.5 bool is_bound ( ) [inline]

Indicates if the ServiceProxy is currently bound to a particular instance_name.

#### 15.9.2.6 bool is_null ( ) [inline]

Indicates if the entity has been 'closed'.

If is_null() is true, then the underlying dds entities have been released and the RPCEntity is no longer useful for operations.

#### 15.9.2.7 void unbind ( ) [inline]

Undo any previous 'bind()' operation.

After this, the ServiceProxy will communicate with any known services.

#### 15.9.2.8 ReturnCode_t wait_for_service ( ) [inline]

Block until any service is discovered.

**Return values**

| | |
|---|---|
| *RETCODE_OK* | : When matched with requested service[s]. |

#### 15.9.2.9 ReturnCode_t wait_for_service ( Duration_t *maxWait* ) [inline]

Block up to 'maxWait' until any service is discovered.

**Return values**

| | |
|---|---|
| *RETCODE_OK* | : If matched with requested service[s]. |
| *RETCODE_TIMEOUT* | : If the maxWait time passes without discovering a service. |

#### 15.9.2.10 ReturnCode_t wait_for_service ( String *instanceName* ) [inline]

Wait for a service with matching 'instanceName' to be discovered.

**Return values**

| | |
|---|---|
| *RETCODE_OK* | : When matched with requested service[s]. |

#### 15.9.2.11 ReturnCode_t wait_for_service ( Duration_t *max_wait,* String *instanceName* ) [inline]

Block up to 'maxWait' until a service matching 'instanceNmae' is discovered.

**Return values**

| | |
|---|---|
| *RETCODE_OK* | : If matched with requested service[s]. |
| *RETCODE_TIMEOUT* | : If the maxWait time passes without discovering a service. |

#### 15.9.2.12 ReturnCode_t wait_for_services ( uint *count* ) [inline]

Block until at least 'count' services have been discovered.

**Return values**

| | |
|---|---|
| *RETCODE_OK* | : When matched with requested service[s]. |

#### 15.9.2.13 ReturnCode_t wait_for_services ( Duration_t *max_wait,* uint *count* ) [inline]

Block until at least 'count' services have been discovered or 'maxWait' time elapses.

**Return values**

| | |
|---|---|
| *RETCODE_OK* | : If matched with requested service[s]. |
| *RETCODE_TIMEOUT* | : If the maxWait time passes without discovering 'count' services. |

#### 15.9.2.14 ReturnCode_t wait_for_services ( List< String > *instanceNames* ) [inline]

Block until discovering a service for each of the listed 'instanceNames'.

**Return values**

| | |
|---|---|
| *RETCODE_OK* | : When matched with requested service[s]. |

#### 15.9.2.15 ReturnCode_t wait_for_services ( Duration_t *max_wait,* List< String > *instanceNames* ) [inline]

Block until discovering a service for each of the listed 'instanceNames', or 'maxWait' elapses.

**Return values**

| | |
|---|---|
| *RETCODE_OK* | : If matched with requested service[s]. |
| *RETCODE_TIMEOUT* | : If the maxWait time passes without discovering the specified services. |

## 15.10 ServiceStatus Enum Reference

### 15.10.1 Description

Indicates the status of a ServiceEndpoint

#### Public Attributes

- CLOSED
- PAUSED
- RUNNING

### 15.10.2 Member Data Documentation

#### 15.10.2.1 CLOSED

The ServiceEndpoint has been closed.

#### 15.10.2.2 PAUSED

The ServiceEndpoint is paused.

#### 15.10.2.3 RUNNING

The ServiceEndpoint is running.

## 15.11 ListenerBase Interface Reference

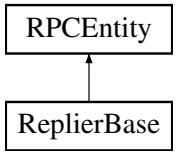
### 15.11.1 Description

Basis for all rpc related listener classes.

## 15.12 ReplierBase Interface Reference

### 15.12.1 Description

Provides a generic basis for all Replier objects. Inheritance diagram for ReplierBase:




#### Additional Inherited Members

## 15.13 ReplierListener< TReq, TRep > Interface Template Reference

### 15.13.1 Description

ReplierLister can be installed on a Replier.

ReplierListener<TReq, TRep> interface is used to provide an asynchronous request listener for a Replier. It is passed to the Replier constructor through ReplierParams. It extends ListenerBase interface and enables asynchronous processing of the requests. The callback is provided the Replier object and returns void.

**See also**

ReplierParams

Inherits ListenerBasewhere TReq RequestType, newwhere TRep ReplyType, and new.

### Public Member Functions

- void on_request_available (Replier< TReq, TRep > replier)

### 15.13.2 Member Function Documentation

#### 15.13.2.1 void on_request_available ( Replier< TReq, TRep > *replier* )

Invoked when the Replier has one or more requests pending.

The method can process the available requests and send replies as required.

## 15.14 RequesterListener< TReq, TRep > Interface Template Reference

### 15.14.1 Description

RequesterListener can be installed on a Requester.

RequesterListener<TReq,TRep> interface is used to receive notification of reply arrival. It is passed to the Requester through RequesterParams. It extends ListenerBase and enables processing of replies. The callback is provided the replier object and returns void.

**See also**

RequesterParams

Inherits ListenerBasewhere TReq RequestType, newwhere TRep ReplyType, and new.

### Public Member Functions

- void on_reply_available (Requester< TReq, TRep > requester)

### 15.14.2 Member Function Documentation

#### 15.14.2.1 void on_reply_available ( Requester< TReq, TRep > *requester* )

Invoked when one or more replies are available.

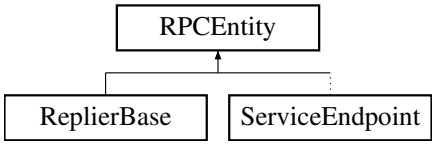
Can process the replies as appropriate.

## 15.15 RPCEntity Interface Reference

### 15.15.1 Description

RPCEntity is a base abstract class.

RPCEntity is the base abstract class extended by all the active RPC entities. It supports close() and is_null() operations.
Inheritance diagram for RPCEntity:




### Public Member Functions

- void close ()
- bool is_null ()

### 15.15.2 Member Function Documentation

#### 15.15.2.1 void close ( )

Release the underlying resources including any DDS entities.

The RPCEntity will no longer be useful and will not participate in any communication.

Implemented in ServiceEndpoint.

#### 15.15.2.2 bool is_null ( )

Indicates if the entity has been 'closed'.

If is_null() is true, then the underlying dds entities have been released and the RPCEntity is no longer useful for operations.

Implemented in ServiceEndpoint.

## 15.16 SimpleReplierListener< TReq, TRep > Interface Template Reference

### 15.16.1 Description

SimpleReplierListener can be installed on a Replier.

SimpleReplierListener<TReq, TRep> is used to provide a synchronous request listener for a Replier. ReplierParams is used to pass an instance of SimpleReplierListener<TReq, TRep>. It extends the ListenerBase abstract class and enables synchronous processing of the requests. The callback is provided the request object and must return a reply.

**See also**

ReplierParams

Inherits ListenerBasewhere TReq RequestType, newwhere TRep ReplyType, and new.

### Public Member Functions

- TRep process_request (Sample< TReq >sample, SampleIdentity_t sample_id)

### 15.16.2 Member Function Documentation

#### 15.16.2.1 TRep process_request ( Sample< TReq > *sample,* SampleIdentity_t *sample_id* )

Invoked when the Replier has recieved a request.

Must initilaize and return a Reply.

## 15.17 SimpleRequesterListener< TRep > Interface Template Reference

### 15.17.1 Description

SimpleRequesterListener can be installed on a Requester.

SimpleRequesterListener<TRep> abstract class is used to receive notifications of arrival of replies. It is passed to the Requester through RequesterParams. It extends ListenerBase and enables processing of replies. The callback is provided access to the reply sample and returns void.

**See also**

RequesterParams

Inherits ListenerBasewhere TRep ReplyType, and new.

### Public Member Functions

- void process_reply (Sample< TRep > sample, SampleIdentity_t sample_identity)

### 15.17.2 Member Function Documentation

#### 15.17.2.1 void process_reply ( Sample< TRep > *sample,* SampleIdentity_t *sample_identity* )

Invoked when a reply is available.

Must process the reply.

# Chapter 16

# CoreDX DDS DynamicTypes

NOTE: This API is deprecated. The preferred DynamicType API is the one provided by the XTypes implementation.

# Chapter 17

# Data Structure Index

## 17.1 Class List

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

# Chapter 18

# Not Yet Supported

**Member DataReader.create_querycondition (uint sample_states, uint view_states, uint instance_states, String query_expression, List< String > query_parameters)**

QueryConditions are not yet supported as triggers for a WaitSet.

**Member DomainParticipant.create_multitopic (String name, String type_name, String subscription_expression, List< String > expression_params)**

This is currently unsupported in the C# language binding.

**Member DomainParticipant.delete_multitopic (MultiTopic a_multitopic)**

This is currently unsupported in the C# language binding.

**Member DomainParticipant.get_discovered_topic_data (TopicBuiltinTopicData topic_data, InstanceHandle_t topic_handle)**

This is currently unsupported in the C# language binding.

**Member DomainParticipant.get_discovered_topics (List< InstanceHandle_t > topic_handles)**

This is currently unsupported in the C# language binding.

**Member DomainParticipant.ignore_publication (InstanceHandle_t handle)**

This is currently unsupported in the C# language binding.

**Member DomainParticipant.ignore_subscription (InstanceHandle_t handle)**

This is currently unsupported in the C# language binding.

**Member DomainParticipant.ignore_topic (InstanceHandle_t handle)**

This is currently unsupported in the C# language binding.

**Class MultiTopic**

CoreDX DDS for C# does not yet implement MultiTopics.

**Member Publisher.begin_coherent_changes ()**

This operation is not yet implemented.

**Member Publisher.end_coherent_changes ()**

This operation is not yet implemented.

**Member Publisher.resume_publications ()**

This operation is not yet implemented.

**Member Publisher.suspend_publications ()**

This operation is not yet implemented.

**Class QueryCondition**

CoreDX DDS does not yet support QueryConditions as triggers for a WaitSet.

**Member Replier< TReq, TRep >.receive_nondata_samples (bool enable)**

**Member Requester< TReq, TRep >.wait_for_replies (int min_count, Duration_t max_wait, SampleIdentity_t related_request_id)**

This routine currently ignores 'related_request_id', and will return with success if *any* 'min_count' replies are available.

**Member Subscriber.begin_access ()**

This is currently unsupported in the C# language binding.

**Member Subscriber.end_access ()**

This is currently unsupported in the C# language binding.

**Member Subscriber.get_datareaders (List< DataReader > readers, long sample_states, long view_states, long instance_states)**

This is currently unsupported in the C# language binding.

# Chapter 19

# Hierarchical Index

## 19.1 Class Hierarchy

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

# Chapter 20

# Namespace Details

## 20.1 Namespace List

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

## 20.2 Package com.toc.coredx.DDS.rpc

### Classes

- class ClientEndpoint
- class ClientParams
- interface ListenerBase
- class Replier
- interface ReplierBase
- interface ReplierListener
- class ReplierParams
- class Requester
- interface RequesterListener
- class RequesterParams
- interface RPCEntity
- class ServiceEndpoint
- class ServiceParams
- class ServiceProxy
- enum ServiceStatus
- interface SimpleReplierListener
- interface SimpleRequesterListener

### 20.2.1 Detailed Description

Provides the RPC over DDS infrastructure

# Index