CoreDX TM Data Distribution Service

The leading Small Footprint DDS Middleware

C++ Reference Manual

Twin Oaks Computing, Inc

Castle Rock, CO 80108

Jan 2017

©2009-2017 Twin Oaks Computing, Inc

All rights reserved.
Published online 2009-2017

Trademarks

Twin Oaks Computing, and CoreDX DDS, and the CoreDX DDS logo are trademarks of Twin Oaks Computing, Inc. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

Copy and Use Restrictions

No part of this document may be reproduced, stored, or transmitted (electronically or mechanically) without the prior written permission of Twin Oaks Computing, Inc. The software documented in this publication is provided pursuant to a License Agreement containing restrictions on its use.

DISCLAIMER OF WARRANTY

THIS DOCUMENT IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

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

1 Introduction **1**

1.1 DDS Entities . . . 3

1.1.1 Detailed Description . . . 3

1.2 DDS Quality of Service . . . 4

1.2.1 Detailed Description . . . 5

1.2.2 Function Documentation . . . 5

1.2.2.1 get_datareader_qos(const char ∗id) . . . 5

1.2.2.2 get_datawriter_qos(const char ∗id) . . . 5

1.2.2.3 get_participant_qos(const char ∗id) . . . 5

1.2.2.4 get_participantfactory_qos(const char ∗id) . . . 5

1.2.2.5 get_publisher_qos(const char ∗id) . . . 5

1.2.2.6 get_subscriber_qos(const char ∗id) . . . 6

1.2.2.7 get_topic_qos(const char ∗id) . . . 6

1.2.2.8 QosProvider(const char ∗uri, const char ∗profile, CDX_XmlApi ∗xml_parser) . . . 6

1.2.2.9 return_datareader_qos(DataReaderQos ∗qos) . . . 6

1.2.2.10 return_datawriter_qos(DataWriterQos ∗qos) . . . 6

1.2.2.11 return_participant_qos(DomainParticipantQos ∗qos) . . . 6

1.2.2.12 return_participantfactory_qos(DomainParticipantFactoryQos ∗qos) . . . 6

1.2.2.13 return_publisher_qos(PublisherQos ∗qos) . . . 6

1.2.2.14 return_subscriber_qos(SubscriberQos ∗qos) . . . 7

1.2.2.15 return_topic_qos(TopicQos ∗qos) . . . 7

1.2.2.16 ∼QosProvider() . . . 7

1.3 DDS Conditions, Listeners, and WaitSets . . . 8

1.3.1 Detailed Description . . . 8

1.4 DDS Listeners . . . 9

1.4.1 Detailed Description . . . 9

1.5 DDS Conditions . . . 10

1.5.1 Detailed Description . . . 10

1.6 DDS WaitSets . . . 11
1.6.1 Detailed Description . . . 11
1.7 CoreDX DDS 'Sample' objects. . . . 12
1.7.1 Detailed Description . . . 12
1.8 DDS Status Structures . . . 13
1.8.1 Detailed Description . . . 13
1.9 CoreDX DDS Transports . . . 15
1.9.1 Detailed Description . . . 15
1.10 CoreDX DDS RPC . . . 16
1.10.1 Detailed Description . . . 17
1.10.2 Enumeration Type Documentation . . . 17
1.10.2.1 ServiceStatus . . . 17
1.11 CoreDX DDS XML API . . . 18
1.11.1 Detailed Description . . . 18
1.12 X-Types Dynamic Types . . . 19
1.12.1 Detailed Description . . . 19
1.13 CoreDX DDS DynamicTypes . . . 20
1.13.1 Detailed Description . . . 21

2 Primary DDS Entities and Types 23

2.1 DDS::Condition Class Reference . . . 23
2.1.1 Detailed Description . . . 23
2.1.2 Member Function Documentation . . . 23
2.1.2.1 get_trigger_value() . . . 23
2.2 DDS::ContentFilteredTopic Class Reference . . . 24
2.2.1 Detailed Description . . . 24
2.2.2 Member Function Documentation . . . 25
2.2.2.1 get_expression_parameters(StringSeq &params) . . . 25
2.2.2.2 get_name() . . . 25
2.2.2.3 get_related_topic() . . . 25
2.2.2.4 get_type_name() . . . 25
2.2.2.5 set_expression_parameters(const StringSeq &params) . . . 25
2.3 DDS::DataReader Class Reference . . . 26
2.3.1 Detailed Description . . . 28
2.3.2 Member Function Documentation . . . 28
2.3.2.1 create_querycondition(SampleStateMask sample_states, ViewStateMask view_states, InstanceStateMask instance_states, const char *query_expression, const StringSeq &query_parameters) . . . 28

2.3.2.2 create_readcondition(SampleStateMask sample_states, ViewStateMask view_states, InstanceStateMask instance_states) . . . 29
2.3.2.3 delete_contained_entities() . . . 29
2.3.2.4 delete_readcondition(ReadCondition *a_condition) . . . 29
2.3.2.5 enable() . . . 29
2.3.2.6 get_key_value(void *key_holder, InstanceHandle_t handle) . . . 30
2.3.2.7 get_listener() . . . 30
2.3.2.8 get_liveliness_changed_status(LivelinessChangedStatus *status) . . . 30
2.3.2.9 get_matched_publication_data(PublicationBuiltinTopicData *publication_data, const InstanceHandle_t publication_handle) . . . 30
2.3.2.10 get_matched_publications(InstanceHandleSeq *publication_handles) . . . 31
2.3.2.11 get_qos(DataReaderQos &qos) . . . 31
2.3.2.12 get_requested_deadline_missed_status(RequestedDeadlineMissedStatus *status) . . . 31
2.3.2.13 get_requested_incompatible_qos_status(RequestedIncompatibleQosStatus *status) . . . 31
2.3.2.14 get_sample_lost_status(SampleLostStatus *status) . . . 31
2.3.2.15 get_sample_rejected_status(SampleRejectedStatus *status) . . . 31
2.3.2.16 get_status_changes() . . . 31
2.3.2.17 get_subscription_matched_status(SubscriptionMatchedStatus *status) . . . 32
2.3.2.18 get_topicdescription() . . . 32
2.3.2.19 lookup_instance(void *instance_data) . . . 32
2.3.2.20 read(Sequence *received_data, SampleInfoSeq *sample_infos, long max_samples, SampleStateMask sample_states, ViewStateMask view_states, InstanceStateMask instance_states) . . . 32
2.3.2.21 read_instance(Sequence *received_data, SampleInfoSeq *sample_infos, long max_samples, InstanceHandle_t a_handle, SampleStateMask sample_states, ViewStateMask view_states, InstanceStateMask instance_states) . . . 33
2.3.2.22 read_next_instance(Sequence *received_data, SampleInfoSeq *sample_infos, long max_samples, InstanceHandle_t previous_handle, SampleStateMask sample_states, ViewStateMask view_states, InstanceStateMask instance_states) . . . 33
2.3.2.23 read_next_instance_w_condition(Sequence *received_data, SampleInfoSeq *sample_infos, long max_samples, InstanceHandle_t previous_handle, ReadCondition *a_condition) . . . 33
2.3.2.24 read_next_sample(void *received_data, SampleInfo *sample_info) . . . 34
2.3.2.25 read_w_condition(Sequence *received_data, SampleInfoSeq *sample_infos, long max_samples, ReadCondition *a_condition) . . . 34
2.3.2.26 return_loan(Sequence *received_data, SampleInfoSeq *sample_infos) . . . 34
2.3.2.27 set_listener(DataReaderListener *a_listener, StatusMask mask) . . . 35
2.3.2.28 set_qos(const DataReaderQos &qos) . . . 35
2.3.2.29 take(Sequence *received_data, SampleInfoSeq *sample_infos, long max_samples, SampleStateMask sample_states, ViewStateMask view_states, InstanceStateMask instance_states) . . . 35

2.3.2.30 take_instance(Sequence *received_data, SampleInfoSeq *sample_infos, long max_samples, InstanceHandle_t a_handle, SampleStateMask sample_states, ViewStateMask view_states, InstanceStateMask instance_states) . . . 36
2.3.2.31 take_next_instance(Sequence *received_data, SampleInfoSeq *sample_infos, long max_samples, InstanceHandle_t previous_handle, SampleStateMask sample_states, ViewStateMask view_states, InstanceStateMask instance_states) . . . 36
2.3.2.32 take_next_instance_w_condition(Sequence *received_data, SampleInfoSeq *sample_infos, long max_samples, InstanceHandle_t previous_handle, ReadCondition *a_condition) . . . 36
2.3.2.33 take_next_sample(void *received_data, SampleInfo *sample_info) . . . 37
2.3.2.34 take_w_condition(Sequence *received_data, SampleInfoSeq *sample_infos, long max_samples, ReadCondition *a_condition) . . . 37
2.4 DDS::DataWriter Class Reference . . . 37
2.4.1 Detailed Description . . . 39
2.4.2 Member Function Documentation . . . 39
2.4.2.1 assert_liveliness() . . . 39
2.4.2.2 dispose(const void *instance_data, const InstanceHandle_t instance_handle) . . . 39
2.4.2.3 dispose_w_timestamp(const void *instance_data, const InstanceHandle_t instance_handle, const Time_t &source_timestamp) . . . 39
2.4.2.4 enable() . . . 40
2.4.2.5 get_key_value(void *key_holder, const InstanceHandle_t handle) . . . 40
2.4.2.6 get_listener() . . . 40
2.4.2.7 get_liveliness_lost_status(LivelinessLostStatus *status) . . . 40
2.4.2.8 get_matched_subscription_data(SubscriptionBuiltinTopicData *subscription_data, InstanceHandle_t &subscription_handle) . . . 40
2.4.2.9 get_matched_subscriptions(InstanceHandleSeq *subscription_handles) . . . 41
2.4.2.10 get_offered_deadline_missed_status(OfferedDeadlineMissedStatus *status) . . . 41
2.4.2.11 get_offered_incompatible_qos_status(OfferedIncompatibleQosStatus *status) . . . 41
2.4.2.12 get_publication_matched_status(PublicationMatchedStatus *status) . . . 41
2.4.2.13 get_qos(DataWriterQos &qos) . . . 41
2.4.2.14 get_status_changes() . . . 41
2.4.2.15 lookup_instance(const void *instance_data) . . . 41
2.4.2.16 register_instance(const void *instance_data) . . . 42
2.4.2.17 register_instance_w_timestamp(const void *instance_data, const Time_t &source_timestamp) 42
2.4.2.18 set_listener(DataWriterListener *a_listener, StatusMask mask) . . . 42
2.4.2.19 set_qos(const DataWriterQos &qos) . . . 42
2.4.2.20 unregister_instance(const void *instance_data, const InstanceHandle_t handle) . . . 42
2.4.2.21 unregister_instance_w_timestamp(const void *instance_data, const InstanceHandle_t handle, const Time_t &source_timestamp) . . . 42
2.4.2.22 wait_for_acknowledgments(const Duration_t &max_wait) . . . 42

2.4.2.23 write(const void ∗instance_data, const InstanceHandle_t handle) . . . 43
2.4.2.24 write_w_timestamp(const void ∗instance_data, const InstanceHandle_t handle, const Time_t &source_timestamp) . . . 43
2.5 DDS::DomainEntity Class Reference . . . 43
2.5.1 Detailed Description . . . 43
2.6 DDS::DomainParticipant Class Reference . . . 44
2.6.1 Detailed Description . . . 46
2.6.2 Member Function Documentation . . . 46
2.6.2.1 assert_liveliness() . . . 46
2.6.2.2 check_version(const char ∗major, const char ∗minor, const char ∗patch) . . . 46
2.6.2.3 contains_entity(InstanceHandle_t a_handle) . . . 47
2.6.2.4 create_contentfilteredtopic(const char ∗name, class Topic ∗related_topic, const char ∗filter_expression, const StringSeq &filter_parameters) . . . 47
2.6.2.5 create_multitopic(const char ∗name, const char ∗type_name, const char ∗subscription_expression, const StringSeq &expression_params) . . . 47
2.6.2.6 create_publisher(const PublisherQos &qos, PublisherListener ∗a_listener, StatusMask mask) . . . 47
2.6.2.7 create_subscriber(const SubscriberQos &qos, SubscriberListener ∗a_listener, StatusMask mask) . . . 48
2.6.2.8 create_topic(const char ∗topic_name, const char ∗type_name, const TopicQos &qos, TopicListener ∗a_listener, StatusMask mask) . . . 48
2.6.2.9 delete_contained_entities() . . . 48
2.6.2.10 delete_contentfilteredtopic(class ContentFilteredTopic ∗a_cft) . . . 48
2.6.2.11 delete_multitopic(class MultiTopic ∗a_multitopic) . . . 48
2.6.2.12 delete_publisher(class Publisher ∗p) . . . 49
2.6.2.13 delete_subscriber(class Subscriber ∗s) . . . 49
2.6.2.14 delete_topic(class Topic ∗a_topic) . . . 49
2.6.2.15 do_work(const Duration_t &time_slice) . . . 49
2.6.2.16 enable() . . . 49
2.6.2.17 find_topic(const char ∗topic_name, const Duration_t &timeout) . . . 50
2.6.2.18 get_builtin_subscriber() . . . 50
2.6.2.19 get_default_publisher_qos(PublisherQos &qos) . . . 50
2.6.2.20 get_default_subscriber_qos(SubscriberQos &qos) . . . 50
2.6.2.21 get_default_topic_qos(TopicQos &qos) . . . 50
2.6.2.22 get_discovered_participant_data(ParticipantBuiltinTopicData ∗participant_data, InstanceHandle_t &participant_handle) . . . 51
2.6.2.23 get_discovered_participants(InstanceHandleSeq ∗participant_handles) . . . 51
2.6.2.24 get_discovered_topic_data(TopicBuiltinTopicData ∗topic_data, InstanceHandle_t &topic_handle) . . . 51

2.6.2.25 get_discovered_topics(InstanceHandleSeq *topic_handles) . . . 51
2.6.2.26 get_listener() . . . 51
2.6.2.27 get_status_changes() . . . 51
2.6.2.28 ignore_participant(InstanceHandle_t &handle) . . . 52
2.6.2.29 ignore_publication(InstanceHandle_t &handle) . . . 52
2.6.2.30 ignore_subscription(InstanceHandle_t &handle) . . . 52
2.6.2.31 ignore_topic(InstanceHandle_t &handle) . . . 52
2.6.2.32 lookup_topicdescription(const char *name) . . . 52
2.6.2.33 register_type(TypeSupport *ts, const char *type_name) . . . 53
2.6.2.34 set_default_publisher_qos(const PublisherQos &qos) . . . 53
2.6.2.35 set_default_subscriber_qos(const SubscriberQos &qos) . . . 53
2.6.2.36 set_default_topic_qos(const TopicQos &qos) . . . 53
2.6.2.37 set_listener(DomainParticipantListener *a_listener, StatusMask mask) . . . 53
2.6.2.38 set_qos(const DomainParticipantQos &qos) . . . 53
2.7 DDS::DomainParticipantFactory Class Reference . . . 54
2.7.1 Detailed Description . . . 54
2.7.2 Member Function Documentation . . . 54
2.7.2.1 create_participant(DomainId_t domain_id, const DomainParticipantQos &qos, DomainParticipantListener *a_listener, StatusMask mask) . . . 54
2.7.2.2 delete_participant(DomainParticipant *a_participant) . . . 55
2.7.2.3 get_default_participant_qos(DomainParticipantQos &qos) . . . 55
2.7.2.4 get_instance() . . . 55
2.7.2.5 lookup_participant(DomainId_t domain_id) . . . 55
2.7.2.6 set_default_participant_qos(const DomainParticipantQos &qos) . . . 55
2.7.2.7 set_qos(const DomainParticipantFactoryQos &qos) . . . 55
2.8 DDS::Entity Class Reference . . . 55
2.8.1 Detailed Description . . . 56
2.8.2 Member Function Documentation . . . 56
2.8.2.1 enable() . . . 56
2.9 DDS::GuardCondition Class Reference . . . 56
2.9.1 Detailed Description . . . 57
2.9.2 Member Function Documentation . . . 57
2.9.2.1 get_trigger_value() . . . 57
2.9.2.2 set_trigger_value(bool v) . . . 57
2.10 DDS::Publisher Class Reference . . . 57
2.10.1 Detailed Description . . . 58
2.10.2 Member Function Documentation . . . 58

2.10.2.1 begin_coherent_changes() . . . 58
2.10.2.2 copy_from_topic_qos(DataWriterQos &a_datawriter_qos, const TopicQos &a_topic_qos) 58
2.10.2.3 create_datawriter(Topic *a_topic, const DataWriterQos &qos, DataWriterListener *a_listener, StatusMask mask) . . . 59
2.10.2.4 delete_contained_entities() . . . 59
2.10.2.5 delete_datawriter(DataWriter *a_datawriter) . . . 59
2.10.2.6 enable() . . . 59
2.10.2.7 end_coherent_changes() . . . 60
2.10.2.8 get_default_datawriter_qos(DataWriterQos &qos) . . . 60
2.10.2.9 get_listener() . . . 60
2.10.2.10 get_qos(PublisherQos &qos) . . . 60
2.10.2.11 get_status_changes() . . . 60
2.10.2.12 lookup_datawriter(const char *topic_name) . . . 60
2.10.2.13 resume_publications() . . . 61
2.10.2.14 set_default_datawriter_qos(const DataWriterQos &qos) . . . 61
2.10.2.15 set_listener(PublisherListener *a_listener, StatusMask mask) . . . 61
2.10.2.16 set_qos(const PublisherQos &qos) . . . 61
2.10.2.17 suspend_publications() . . . 61
2.10.2.18 wait_for_acknowledgments(const Duration_t &max_wait) . . . 61
2.11 DDS::QueryCondition Class Reference . . . 61
2.11.1 Detailed Description . . . 62
2.11.2 Member Function Documentation . . . 62
2.11.2.1 get_query_expression(void) . . . 62
2.11.2.2 get_query_parameters(StringSeq &seq) . . . 62
2.11.2.3 set_query_parameters(const StringSeq &seq) . . . 63
2.12 DDS::ReadCondition Class Reference . . . 63
2.12.1 Detailed Description . . . 63
2.12.2 Member Function Documentation . . . 64
2.12.2.1 get_trigger_value() . . . 64
2.13 DDS::StatusCondition Class Reference . . . 64
2.13.1 Detailed Description . . . 64
2.13.2 Member Function Documentation . . . 64
2.13.2.1 get_enabled_statuses() . . . 64
2.13.2.2 get_trigger_value() . . . 65
2.13.2.3 set_enabled_statuses(StatusMask mask) . . . 65
2.14 DDS::Subscriber Class Reference . . . 65
2.14.1 Detailed Description . . . 66

2.14.2 Member Function Documentation . . . 66
2.14.2.1 begin_access() . . . 66
2.14.2.2 copy_from_topic_qos(DataReaderQos &a_datareader_qos, const TopicQos &a_topic_qos) . . . 66
2.14.2.3 create_datareader(TopicDescription *a_topic, const DataReaderQos &qos, DataReaderListener *a_listener, StatusMask mask) . . . 67
2.14.2.4 delete_contained_entities() . . . 67
2.14.2.5 delete_datareader(class DataReader *a_datareader) . . . 67
2.14.2.6 enable() . . . 67
2.14.2.7 get_datareaders(DataReaderSeq *readers, SampleStateMask sample_states, ViewStateMask view_states, InstanceStateMask instance_states) . . . 68
2.14.2.8 get_default_datareader_qos(DataReaderQos &qos) . . . 68
2.14.2.9 get_listener() . . . 68
2.14.2.10 get_qos(SubscriberQos &qos) . . . 68
2.14.2.11 get_status_changes() . . . 69
2.14.2.12 lookup_datareader(const char *topic_name) . . . 69
2.14.2.13 set_default_datareader_qos(const DataReaderQos &qos) . . . 69
2.14.2.14 set_listener(SubscriberListener *a_listener, StatusMask mask) . . . 69
2.14.2.15 set_qos(const SubscriberQos &qos) . . . 69
2.15 DDS::Topic Class Reference . . . 69
2.15.1 Detailed Description . . . 70
2.15.2 Member Function Documentation . . . 70
2.15.2.1 enable() . . . 70
2.15.2.2 get_inconsistent_topic_status(InconsistentTopicStatus *a_status) . . . 71
2.15.2.3 get_listener() . . . 71
2.15.2.4 get_qos(TopicQos &qos) . . . 71
2.15.2.5 get_status_changes() . . . 71
2.15.2.6 set_listener(TopicListener *a_listener, StatusMask mask) . . . 71
2.15.2.7 set_qos(const TopicQos &qos) . . . 72
2.16 DDS::TopicDescription Class Reference . . . 72
2.16.1 Detailed Description . . . 72
2.16.2 Member Function Documentation . . . 72
2.16.2.1 get_name() . . . 72
2.16.2.2 get_type_name() . . . 73
2.17 DDS::WaitSet Class Reference . . . 73
2.17.1 Detailed Description . . . 73
2.17.2 Member Function Documentation . . . 73
2.17.2.1 attach_condition(Condition &a_condition) . . . 73

2.17.2.2 detach_condition(Condition &a_condition) . . . 73
2.17.2.3 get_conditions(ConditionSeq &attached_conditions) . . . 74
2.17.2.4 wait(ConditionSeq &active_conditions, const Duration_t &timeout) . . . 74

3 Entity QoS Collections **75**
3.1 DDS::DataReaderQos Class Reference . . . 75
3.1.1 Detailed Description . . . 76
3.2 DDS::DataWriterQos Class Reference . . . 76
3.2.1 Detailed Description . . . 77
3.3 DDS::DomainParticipantFactoryQos Class Reference . . . 77
3.3.1 Detailed Description . . . 78
3.4 DDS::DomainParticipantQos Class Reference . . . 78
3.4.1 Detailed Description . . . 78
3.5 DDS::PublisherQos Class Reference . . . 78
3.5.1 Detailed Description . . . 79
3.5.2 Member Data Documentation . . . 79
3.5.2.1 partition . . . 79
3.5.2.2 presentation . . . 79
3.6 DDS::SubscriberQos Class Reference . . . 79
3.6.1 Detailed Description . . . 80
3.6.2 Member Data Documentation . . . 80
3.6.2.1 partition . . . 80
3.6.2.2 presentation . . . 80
3.7 DDS::TopicQos Class Reference . . . 80
3.7.1 Detailed Description . . . 81

4 QoS Policies **83**
4.1 DDS::DataRepresentationQosPolicy Struct Reference . . . 83
4.1.1 Detailed Description . . . 83
4.1.2 Constructor & Destructor Documentation . . . 83
4.1.2.1 DataRepresentationQosPolicy() . . . 83
4.2 DDS::DeadlineQosPolicy Struct Reference . . . 83
4.2.1 Detailed Description . . . 84
4.2.2 Constructor & Destructor Documentation . . . 84
4.2.2.1 DeadlineQosPolicy() . . . 84
4.3 DDS::DestinationOrderQosPolicy Struct Reference . . . 84
4.3.1 Detailed Description . . . 84
4.3.2 Constructor & Destructor Documentation . . . 85

4.3.2.1 DestinationOrderQosPolicy() . . . 85
4.4 DiscoveryQosPolicy Struct Reference . . . 85
4.4.1 Detailed Description . . . 85
4.5 DDS::DurabilityQosPolicy Struct Reference . . . 85
4.5.1 Detailed Description . . . 86
4.5.2 Constructor & Destructor Documentation . . . 86
4.5.2.1 DurabilityQosPolicy() . . . 86
4.6 DDS::DurabilityServiceQosPolicy Struct Reference . . . 86
4.6.1 Detailed Description . . . 87
4.6.2 Constructor & Destructor Documentation . . . 87
4.6.2.1 DurabilityServiceQosPolicy() . . . 87
4.7 EntityFactoryQosPolicy Struct Reference . . . 87
4.7.1 Detailed Description . . . 87
4.8 EntityNameQosPolicy Struct Reference . . . 88
4.8.1 Detailed Description . . . 88
4.9 DDS::GroupDataQosPolicy Struct Reference . . . 88
4.9.1 Detailed Description . . . 88
4.9.2 Constructor & Destructor Documentation . . . 88
4.9.2.1 GroupDataQosPolicy() . . . 88
4.10 DDS::HistoryQosPolicy Struct Reference . . . 88
4.10.1 Detailed Description . . . 89
4.10.2 Constructor & Destructor Documentation . . . 89
4.10.2.1 HistoryQosPolicy() . . . 89
4.11 DDS::LatencyBudgetQosPolicy Struct Reference . . . 89
4.11.1 Detailed Description . . . 90
4.11.2 Constructor & Destructor Documentation . . . 90
4.11.2.1 LatencyBudgetQosPolicy() . . . 90
4.12 DDS::LifespanQosPolicy Struct Reference . . . 90
4.12.1 Detailed Description . . . 90
4.12.2 Constructor & Destructor Documentation . . . 90
4.12.2.1 LifespanQosPolicy() . . . 90
4.13 DDS::LivelinessQosPolicy Struct Reference . . . 90
4.13.1 Detailed Description . . . 91
4.13.2 Constructor & Destructor Documentation . . . 91
4.13.2.1 LivelinessQosPolicy() . . . 91
4.14 LoggingQosPolicy Struct Reference . . . 91
4.14.1 Detailed Description . . . 92

4.15 DDS::OwnershipQosPolicy Struct Reference . . . 92
4.15.1 Detailed Description . . . 92
4.15.2 Constructor & Destructor Documentation . . . 92
4.15.2.1 OwnershipQosPolicy() . . . 92
4.16 DDS::OwnershipStrengthQosPolicy Struct Reference . . . 92
4.16.1 Detailed Description . . . 93
4.16.2 Constructor & Destructor Documentation . . . 93
4.16.2.1 OwnershipStrengthQosPolicy() . . . 93
4.17 DDS::PartitionQosPolicy Struct Reference . . . 93
4.17.1 Detailed Description . . . 93
4.17.2 Constructor & Destructor Documentation . . . 94
4.17.2.1 PartitionQosPolicy() . . . 94
4.18 PeerParticipantQosPolicy Struct Reference . . . 94
4.18.1 Detailed Description . . . 94
4.18.2 Member Data Documentation . . . 94
4.18.2.1 strict_match . . . 94
4.19 DDS::PresentationQosPolicy Struct Reference . . . 94
4.19.1 Detailed Description . . . 95
4.19.2 Constructor & Destructor Documentation . . . 95
4.19.2.1 PresentationQosPolicy() . . . 95
4.20 DDS::PropertyQosPolicy Struct Reference . . . 95
4.20.1 Detailed Description . . . 95
4.20.2 Constructor & Destructor Documentation . . . 95
4.20.2.1 PropertyQosPolicy() . . . 95
4.21 ReaderDataLifecycleQosPolicy Struct Reference . . . 95
4.21.1 Detailed Description . . . 96
4.22 DDS::ReliabilityQosPolicy Struct Reference . . . 96
4.22.1 Detailed Description . . . 96
4.22.2 Constructor & Destructor Documentation . . . 96
4.22.2.1 ReliabilityQosPolicy() . . . 96
4.23 DDS::ResourceLimitsQosPolicy Struct Reference . . . 97
4.23.1 Detailed Description . . . 97
4.23.2 Constructor & Destructor Documentation . . . 97
4.23.2.1 ResourceLimitsQosPolicy() . . . 97
4.24 DDS::RpcQosPolicy Struct Reference . . . 97
4.24.1 Detailed Description . . . 98
4.24.2 Constructor & Destructor Documentation . . . 98

4.24.2.1 RpcQosPolicy() . . . 98
4.25 RTPSReaderQosPolicy Struct Reference . . . 98
4.25.1 Detailed Description . . . 98
4.26 RTPSWriterQosPolicy Struct Reference . . . 98
4.26.1 Detailed Description . . . 99
4.27 ThreadModelQosPolicy Struct Reference . . . 99
4.27.1 Detailed Description . . . 99
4.28 DDS::TimeBasedFilterQosPolicy Struct Reference . . . 99
4.28.1 Detailed Description . . . 100
4.28.2 Constructor & Destructor Documentation . . . 100
4.28.2.1 TimeBasedFilterQosPolicy() . . . 100
4.29 DDS::TopicDataQosPolicy Struct Reference . . . 100
4.29.1 Detailed Description . . . 100
4.29.2 Constructor & Destructor Documentation . . . 101
4.29.2.1 TopicDataQosPolicy() . . . 101
4.30 DDS::TransportPriorityQosPolicy Struct Reference . . . 101
4.30.1 Detailed Description . . . 101
4.30.2 Constructor & Destructor Documentation . . . 101
4.30.2.1 TransportPriorityQosPolicy() . . . 101
4.31 DDS::TypecodeQosPolicy Struct Reference . . . 101
4.31.1 Detailed Description . . . 102
4.31.2 Constructor & Destructor Documentation . . . 102
4.31.2.1 TypecodeQosPolicy() . . . 102
4.32 DDS::TypeConsistencyEnforcementQosPolicy Struct Reference . . . 102
4.32.1 Detailed Description . . . 102
4.32.2 Constructor & Destructor Documentation . . . 102
4.32.2.1 TypeConsistencyEnforcementQosPolicy() . . . 102
4.33 DDS::UserDataQosPolicy Struct Reference . . . 102
4.33.1 Detailed Description . . . 103
4.33.2 Constructor & Destructor Documentation . . . 103
4.33.2.1 UserDataQosPolicy() . . . 103
4.34 WriterDataLifecycleQosPolicy Struct Reference . . . 103
4.34.1 Detailed Description . . . 103

5 QoS Provider **105**
5.1 DDS::QosProvider Class Reference . . . 105
5.1.1 Detailed Description . . . 106

6 DDS Listeners **107**

6.1 DDS::DataReaderListener Class Reference . . . 107

6.1.1 Detailed Description . . . 107

6.1.2 Member Function Documentation . . . 108

6.1.2.1 on_data_available(DataReader ∗) . . . 108

6.1.2.2 on_liveliness_changed(DataReader ∗, const LivelinessChangedStatus &) . . . 108

6.1.2.3 on_requested_deadline_missed(DataReader ∗, const RequestedDeadlineMissedStatus &) . . . 108

6.1.2.4 on_requested_incompatible_qos(DataReader ∗, const RequestedIncompatibleQosStatus &) . . . 108

6.1.2.5 on_sample_lost(DataReader ∗, const SampleLostStatus &) . . . 108

6.1.2.6 on_sample_rejected(DataReader ∗, const SampleRejectedStatus &) . . . 108

6.1.2.7 on_subscription_matched(DataReader ∗, const SubscriptionMatchedStatus &) . . . 108

6.2 DDS::DataWriterListener Class Reference . . . 109

6.2.1 Detailed Description . . . 109

6.2.2 Member Function Documentation . . . 109

6.2.2.1 on_liveliness_lost(DataWriter ∗, const LivelinessLostStatus &) . . . 109

6.2.2.2 on_offered_deadline_missed(DataWriter ∗, const OfferedDeadlineMissedStatus &) . . . 109

6.2.2.3 on_offered_incompatible_qos(DataWriter ∗, const OfferedIncompatibleQosStatus &) . . . 110

6.2.2.4 on_publication_matched(DataWriter ∗, const PublicationMatchedStatus &) . . . 110

6.3 DDS::DomainParticipantListener Class Reference . . . 110

6.3.1 Detailed Description . . . 110

6.4 DDS::PublisherListener Class Reference . . . 110

6.4.1 Detailed Description . . . 111

6.5 DDS::SubscriberListener Class Reference . . . 111

6.5.1 Detailed Description . . . 111

6.5.2 Member Function Documentation . . . 112

6.5.2.1 on_data_on_readers(Subscriber ∗) . . . 112

6.6 DDS::TopicListener Class Reference . . . 112

6.6.1 Detailed Description . . . 112

6.6.2 Member Function Documentation . . . 112

6.6.2.1 on_inconsistent_topic(Topic ∗, const InconsistentTopicStatus &) . . . 112

7 DDS Status Structures **113**

7.1 InconsistentTopicStatus Struct Reference . . . 113

7.1.1 Detailed Description . . . 113

7.2 LivelinessChangedStatus Struct Reference . . . 113

7.2.1 Detailed Description . . . 114

7.3 LivelinessLostStatus Struct Reference . . . 114
7.3.1 Detailed Description . . . 114
7.4 OfferedDeadlineMissedStatus Struct Reference . . . 114
7.4.1 Detailed Description . . . 115
7.5 OfferedIncompatibleQosStatus Struct Reference . . . 115
7.5.1 Detailed Description . . . 115
7.6 PublicationMatchedStatus Struct Reference . . . 115
7.6.1 Detailed Description . . . 115
7.7 RequestedDeadlineMissedStatus Struct Reference . . . 116
7.7.1 Detailed Description . . . 116
7.8 RequestedIncompatibleQosStatus Struct Reference . . . 116
7.8.1 Detailed Description . . . 116
7.9 SampleLostStatus Struct Reference . . . 116
7.9.1 Detailed Description . . . 117
7.10 SampleRejectedStatus Struct Reference . . . 117
7.10.1 Detailed Description . . . 117
7.11 SubscriptionMatchedStatus Struct Reference . . . 117
7.11.1 Detailed Description . . . 118

8 DDS Samples **119**

8.1 DDS::LoanedSamples< T > Class Template Reference . . . 119
8.1.1 Detailed Description . . . 120
8.1.2 Member Typedef Documentation . . . 121
8.1.2.1 const_iterator . . . 121
8.1.2.2 const_value_type . . . 121
8.1.2.3 difference_type . . . 121
8.1.2.4 iterator . . . 121
8.1.2.5 value_type . . . 121
8.1.3 Member Function Documentation . . . 121
8.1.3.1 move_construct_from_loans(DataReader reader, Seq &data_seq, DDS::SampleInfoSeq &info_seq) . . . 121
8.1.3.2 release(DataReader &reader_ptr, Seq &data_seq, DDS::SampleInfoSeq &info_seq) . 121
8.1.3.3 return_loan() . . . 122
8.1.3.4 to_shared() . . . 122
8.2 DDS::Sample< T > Class Template Reference . . . 122
8.2.1 Detailed Description . . . 123
8.3 DDS::SampleIterator< T, IsConst > Class Template Reference . . . 123
8.3.1 Detailed Description . . . 125

8.3.2 Member Typedef Documentation . . . 125
8.3.2.1 difference_type . . . 125
8.3.2.2 value_type . . . 125
8.4 DDS::SampleRef< T > Class Template Reference . . . 125
8.4.1 Detailed Description . . . 126
8.5 DDS::SharedSamples< T > Class Template Reference . . . 126
8.5.1 Detailed Description . . . 128
8.5.2 Member Typedef Documentation . . . 128
8.5.2.1 const_iterator . . . 128
8.5.2.2 const_value_type . . . 128
8.5.2.3 difference_type . . . 128
8.5.2.4 iterator . . . 128
8.5.2.5 value_type . . . 128
8.6 DDS::WriteSample< T > Class Template Reference . . . 128
8.6.1 Detailed Description . . . 129
8.7 DDS::WriteSampleRef< T > Class Template Reference . . . 129
8.7.1 Detailed Description . . . 130

9 Builtin DDS Types 131

9.1 DDS::BuiltinTopicKey_t Struct Reference . . . 131
9.1.1 Detailed Description . . . 131
9.1.2 Constructor & Destructor Documentation . . . 131
9.1.2.1 BuiltinTopicKey_t() . . . 131
9.1.3 Member Data Documentation . . . 131
9.1.3.1 value . . . 131
9.2 DDS::Duration_t Struct Reference . . . 131
9.2.1 Detailed Description . . . 132
9.2.2 Constructor & Destructor Documentation . . . 132
9.2.2.1 Duration_t() . . . 132
9.3 DDS::GUID_t Struct Reference . . . 132
9.3.1 Detailed Description . . . 132
9.3.2 Constructor & Destructor Documentation . . . 132
9.3.2.1 GUID_t() . . . 132
9.3.3 Member Data Documentation . . . 132
9.3.3.1 value . . . 132
9.4 DDS::Property_t Struct Reference . . . 132
9.4.1 Detailed Description . . . 133

9.4.2 Constructor & Destructor Documentation . . . 133
9.4.2.1 Property_t() . . . 133
9.5 DDS::SampleIdentity_t Struct Reference . . . 133
9.5.1 Detailed Description . . . 133
9.5.2 Constructor & Destructor Documentation . . . 133
9.5.2.1 SampleIdentity_t() . . . 133
9.5.3 Member Data Documentation . . . 134
9.5.3.1 guid . . . 134
9.6 DDS::SequenceNumber_t Struct Reference . . . 134
9.6.1 Detailed Description . . . 134
9.6.2 Constructor & Destructor Documentation . . . 134
9.6.2.1 SequenceNumber_t() . . . 134
9.6.3 Member Data Documentation . . . 134
9.6.3.1 high . . . 134
9.6.3.2 low . . . 134
9.7 DDS::Time_t Struct Reference . . . 134
9.7.1 Detailed Description . . . 134
9.8 DDS_Time_t Struct Reference . . . 135
9.8.1 Detailed Description . . . 135
9.8.2 Member Data Documentation . . . 135
9.8.2.1 nanosec . . . 135
9.8.2.2 sec . . . 135

10 DDS Discovery Types **137**
10.1 DDS::DCPSParticipant Struct Reference . . . 137
10.1.1 Detailed Description . . . 137
10.1.2 Constructor & Destructor Documentation . . . 137
10.1.2.1 DCPSParticipant() . . . 137
10.2 DDS::DCPSPublication Struct Reference . . . 137
10.2.1 Detailed Description . . . 138
10.2.2 Constructor & Destructor Documentation . . . 139
10.2.2.1 DCPSPublication() . . . 139
10.3 DDS::DCPSSubscription Struct Reference . . . 139
10.3.1 Detailed Description . . . 140
10.3.2 Constructor & Destructor Documentation . . . 140
10.3.2.1 DCPSSubscription() . . . 140

11 Supporting Types **141**

11.1 CacheStatistics Struct Reference . . . 141
11.1.1 Detailed Description . . . 141
11.1.2 Member Data Documentation . . . 141
11.1.2.1 instance_alive_count . . . 142
11.1.2.2 instance_count . . . 142
11.1.2.3 instance_disposed_count . . . 142
11.1.2.4 instance_new_count . . . 142
11.1.2.5 instance_no_writers_count . . . 142
11.1.2.6 sample_count . . . 142
11.1.2.7 sample_read_count . . . 142
11.1.2.8 state_flags . . . 142
11.2 Locator Struct Reference . . . 142
11.2.1 Detailed Description . . . 142
11.3 ParticipantLocator Struct Reference . . . 142
11.3.1 Detailed Description . . . 143
11.4 QosPolicyCount Struct Reference . . . 143
11.4.1 Detailed Description . . . 143
11.4.2 Member Data Documentation . . . 143
11.4.2.1 count . . . 143
11.4.2.2 policy_id . . . 143
11.5 SampleInfo Struct Reference . . . 143
11.5.1 Detailed Description . . . 144
11.5.2 Member Data Documentation . . . 144
11.5.2.1 absolute_generation_rank . . . 144
11.5.2.2 generation_rank . . . 145
11.5.2.3 instance_state . . . 145
11.5.2.4 sample_state . . . 145
11.5.2.5 valid_data . . . 145
11.5.2.6 view_state . . . 145
11.6 DDS::sequence< T > Class Template Reference . . . 145
11.6.1 Detailed Description . . . 147
11.6.2 Member Function Documentation . . . 147
11.6.2.1 at(uint32_t n) . . . 147
11.6.2.2 at(uint32_t n) const . . . 147
11.6.2.3 back() . . . 147
11.6.2.4 back() const . . . 147
11.6.2.5 buffer() const . . . 147

11.6.2.6 clear() . . . 147
11.6.2.7 front() . . . 147
11.6.2.8 front() const . . . 147
11.6.2.9 operator[](uint32_t n) . . . 148
11.6.2.10 operator[](uint32_t n) const . . . 148
11.6.2.11 pop_back() . . . 148
11.6.2.12 push_back(const T &v) . . . 148
11.6.2.13 reserve(uint32_t n) . . . 148
11.6.2.14 resize(uint32_t n) . . . 148
11.6.2.15 swap(sequence &x) . . . 148

12 X-Types DynamicType API **149**

12.1 DDS::AnnotationDescriptor Class Reference . . . 149
12.1.1 Detailed Description . . . 149
12.1.2 Constructor & Destructor Documentation . . . 149
12.1.2.1 AnnotationDescriptor() . . . 149
12.1.3 Member Function Documentation . . . 150
12.1.3.1 copy_from(const class AnnotationDescriptor ∗other) . . . 150
12.1.3.2 equals(const class AnnotationDescriptor ∗other) . . . 150
12.1.3.3 get_all_value(Parameters ∗value) . . . 150
12.1.3.4 get_value(char ∗∗value, const char ∗key) . . . 150
12.1.3.5 is_consistent() . . . 150
12.1.3.6 set_value(const char ∗key, const char ∗value) . . . 150
12.1.4 Member Data Documentation . . . 151
12.1.4.1 type . . . 151
12.2 DDS::DynamicData Class Reference . . . 151
12.2.1 Detailed Description . . . 153
12.2.2 Constructor & Destructor Documentation . . . 154
12.2.2.1 DynamicData() . . . 154
12.2.3 Member Function Documentation . . . 155
12.2.3.1 clear_all_values()=0 . . . 155
12.2.3.2 clear_nonkey_values()=0 . . . 155
12.2.3.3 clear_value(const MemberId id)=0 . . . 155
12.2.3.4 clone()=0 . . . 155
12.2.3.5 equals(const class DynamicData ∗other)=0 . . . 156
12.2.3.6 get_boolean_value(unsigned char ∗value, const MemberId id)=0 . . . 156
12.2.3.7 get_boolean_values(BooleanSeq ∗value, const MemberId id)=0 . . . 156

12.2.3.8 get_byte_value(unsigned char ∗value, const MemberId id)=0 156
12.2.3.9 get_byte_values(ByteSeq ∗value, const MemberId id)=0 157
12.2.3.10 get_char32_value(cdx_char32_t ∗value, const MemberId id)=0 157
12.2.3.11 get_char32_values(WcharSeq ∗value, const MemberId id)=0 157
12.2.3.12 get_char8_value(char ∗value, const MemberId id)=0 157
12.2.3.13 get_char8_values(CharSeq ∗value, const MemberId id)=0 157
12.2.3.14 get_complex_value(class DynamicData ∗∗value, const MemberId id)=0 157
12.2.3.15 get_descriptor(class MemberDescriptor ∗value, const MemberId id)=0 158
12.2.3.16 get_float32_value(float ∗value, const MemberId id)=0 158
12.2.3.17 get_float32_values(Float32Seq ∗value, const MemberId id)=0 158
12.2.3.18 get_float64_value(double ∗value, const MemberId id)=0 158
12.2.3.19 get_float64_values(Float64Seq ∗value, const MemberId id)=0 158
12.2.3.20 get_int16_value(short ∗value, const MemberId id)=0 159
12.2.3.21 get_int16_values(Int16Seq ∗value, const MemberId id)=0 159
12.2.3.22 get_int32_value(int ∗value, const MemberId id)=0 159
12.2.3.23 get_int32_values(Int32Seq ∗value, const MemberId id)=0 159
12.2.3.24 get_int64_value(int64_t ∗value, const MemberId id)=0 159
12.2.3.25 get_int64_values(Int64Seq ∗value, const MemberId id)=0 160
12.2.3.26 get_item_count()=0 160
12.2.3.27 get_string_value(char ∗∗value, const MemberId id)=0 160
12.2.3.28 get_string_values(StringSeq ∗value, const MemberId id)=0 160
12.2.3.29 get_uint16_value(unsigned short ∗value, const MemberId id)=0 160
12.2.3.30 get_uint16_values(UInt16Seq ∗value, const MemberId id)=0 161
12.2.3.31 get_uint32_value(unsigned int ∗value, const MemberId id)=0 161
12.2.3.32 get_uint32_values(UInt32Seq ∗value, const MemberId id)=0 161
12.2.3.33 get_uint64_value(uint64_t ∗value, const MemberId id)=0 161
12.2.3.34 get_uint64_values(UInt64Seq ∗value, const MemberId id)=0 161
12.2.3.35 get_wstring_value(cdx_char32_t ∗∗value, const MemberId id)=0 162
12.2.3.36 get_wstring_values(WstringSeq ∗value, const MemberId id)=0 162
12.2.3.37 loan_value(const MemberId id)=0 162
12.2.3.38 return_loaned_value(const class DynamicData ∗value)=0 162
12.2.3.39 set_boolean_value(const MemberId id, const unsigned char value)=0 162
12.2.3.40 set_boolean_values(const MemberId id, const BooleanSeq ∗value)=0 163
12.2.3.41 set_byte_value(const MemberId id, const unsigned char value)=0 163
12.2.3.42 set_byte_values(const MemberId id, const ByteSeq ∗value)=0 163
12.2.3.43 set_char32_value(const MemberId id, const cdx_char32_t value)=0 163
12.2.3.44 set_char32_values(const MemberId id, const WcharSeq ∗value)=0 163

12.2.3.45 set_char8_value(const MemberId id, const char value)=0 . . . 163
12.2.3.46 set_char8_values(const MemberId id, const CharSeq *value)=0 . . . 164
12.2.3.47 set_complex_value(const MemberId id, class DynamicData *value)=0 . . . 164
12.2.3.48 set_float32_value(const MemberId id, const float value)=0 . . . 164
12.2.3.49 set_float32_values(const MemberId id, const Float32Seq *value)=0 . . . 164
12.2.3.50 set_float64_value(const MemberId id, const double value)=0 . . . 164
12.2.3.51 set_float64_values(const MemberId id, const Float64Seq *value)=0 . . . 165
12.2.3.52 set_int16_value(const MemberId id, const short value)=0 . . . 165
12.2.3.53 set_int16_values(const MemberId id, const Int16Seq *value)=0 . . . 165
12.2.3.54 set_int32_value(const MemberId id, const int value)=0 . . . 165
12.2.3.55 set_int32_values(const MemberId id, const Int32Seq *value)=0 . . . 165
12.2.3.56 set_int64_value(const MemberId id, const int64_t value)=0 . . . 166
12.2.3.57 set_int64_values(const MemberId id, const Int64Seq *value)=0 . . . 166
12.2.3.58 set_string_value(const MemberId id, const char *value)=0 . . . 166
12.2.3.59 set_string_values(const MemberId id, const StringSeq *value)=0 . . . 166
12.2.3.60 set_uint16_value(const MemberId id, const unsigned short value)=0 . . . 166
12.2.3.61 set_uint16_values(const MemberId id, const UInt16Seq *value)=0 . . . 167
12.2.3.62 set_uint32_value(const MemberId id, const unsigned int value)=0 . . . 167
12.2.3.63 set_uint32_values(const MemberId id, const UInt32Seq *value)=0 . . . 167
12.2.3.64 set_uint64_value(const MemberId id, const uint64_t value)=0 . . . 167
12.2.3.65 set_uint64_values(const MemberId id, const UInt64Seq *value)=0 . . . 167
12.2.3.66 set_wstring_value(const MemberId id, const cdx_char32_t *value)=0 . . . 167
12.2.3.67 set_wstring_values(const MemberId id, const WstringSeq *value)=0 . . . 168
12.3 DDS::DynamicDataFactory Class Reference . . . 168
12.3.1 Detailed Description . . . 168
12.3.2 Constructor & Destructor Documentation . . . 168
12.3.2.1 DynamicDataFactory() . . . 168
12.3.3 Member Function Documentation . . . 169
12.3.3.1 create_data(const class DynamicType *type) . . . 169
12.4 DDS::DynamicDataReader Class Reference . . . 169
12.4.1 Detailed Description . . . 169
12.5 DDS::DynamicDataWriter Class Reference . . . 169
12.5.1 Detailed Description . . . 170
12.6 DDS::DynamicType Class Reference . . . 170
12.6.1 Detailed Description . . . 170
12.6.2 Constructor & Destructor Documentation . . . 171
12.6.2.1 DynamicType() . . . 171

12.6.3 Member Function Documentation . . . 171
12.6.3.1 equals(const class DynamicType *other)=0 . . . 171
12.6.3.2 get_all_members(DynamicTypeMembersById *member) const =0 . . . 171
12.6.3.3 get_all_members_by_name(DynamicTypeMembersByName *member) const =0 . . . 171
12.6.3.4 get_annotation(class AnnotationDescriptor *descriptor, const unsigned int *idx)=0 . . . 172
12.6.3.5 get_annotation_count()=0 . . . 172
12.6.3.6 get_descriptor(class TypeDescriptor *descriptor) const =0 . . . 172
12.6.3.7 get_kind()=0 . . . 172
12.6.3.8 get_member(class DynamicTypeMember **member, const MemberId id) const =0 . . . 172
12.6.3.9 get_member_by_name(class DynamicTypeMember **member, const char *name) const =0 . . . 173
12.6.3.10 get_name()=0 . . . 173
12.7 DDS::DynamicTypeBuilder Class Reference . . . 173
12.7.1 Detailed Description . . . 174
12.7.2 Constructor & Destructor Documentation . . . 174
12.7.2.1 DynamicTypeBuilder() . . . 174
12.7.3 Member Function Documentation . . . 174
12.7.3.1 add_member(const class MemberDescriptor *descriptor)=0 . . . 174
12.7.3.2 apply_annotation(const class AnnotationDescriptor *descriptor)=0 . . . 175
12.7.3.3 build()=0 . . . 175
12.7.3.4 delete_type(class DynamicType *type)=0 . . . 175
12.7.3.5 equals(const class DynamicType *other)=0 . . . 175
12.7.3.6 get_all_members(DynamicTypeMembersById *member)=0 . . . 176
12.7.3.7 get_all_members_by_name(DynamicTypeMembersByName *member)=0 . . . 176
12.7.3.8 get_annotation(class AnnotationDescriptor *descriptor, const unsigned int idx)=0 . . . 176
12.7.3.9 get_annotation_count()=0 . . . 176
12.7.3.10 get_descriptor(class TypeDescriptor *descriptor)=0 . . . 177
12.7.3.11 get_kind()=0 . . . 177
12.7.3.12 get_member(class DynamicTypeMember **member, const MemberId id)=0 . . . 177
12.7.3.13 get_member_by_name(class DynamicTypeMember **member, const char *name)=0 177
12.7.3.14 get_name()=0 . . . 178
12.8 DDS::DynamicTypeBuilderFactory Class Reference . . . 178
12.8.1 Detailed Description . . . 179
12.8.2 Constructor & Destructor Documentation . . . 179
12.8.2.1 DynamicTypeBuilderFactory() . . . 179
12.8.3 Member Function Documentation . . . 180
12.8.3.1 create_alias_type(const DynamicType *base_type)=0 . . . 180

12.8.3.2 create_array_type(const class DynamicType *element_type, const BoundSeq *bound)=0 . . . 180
12.8.3.3 create_bitset_type(const unsigned int bound)=0 . . . 180
12.8.3.4 create_enumeration_type(const unsigned int bound=32)=0 . . . 181
12.8.3.5 create_map_type(const class DynamicType *key_element_type, const class DynamicType *element_type, const unsigned int bound)=0 . . . 181
12.8.3.6 create_sequence_type(const class DynamicType *element_type, const unsigned int bound)=0 . . . 181
12.8.3.7 create_string_type(const unsigned int bound)=0 . . . 182
12.8.3.8 create_structure_type()=0 . . . 182
12.8.3.9 create_type(const class TypeDescriptor *descriptor)=0 . . . 182
12.8.3.10 create_type_copy(const class DynamicType *type)=0 . . . 182
12.8.3.11 create_type_w_document(const char *document, const char *type_name, const IncludePathSeq *include_paths)=0 . . . 183
12.8.3.12 create_type_w_type_object(const struct TypeObject *type_object)=0 . . . 183
12.8.3.13 create_type_w_uri(const char *document_url, const char *type_name, const IncludePathSeq *include_paths)=0 . . . 183
12.8.3.14 create_union_type()=0 . . . 184
12.8.3.15 create_wstring_type(const unsigned int bound)=0 . . . 184
12.8.3.16 delete_instance() . . . 185
12.8.3.17 delete_type(class DynamicType *type)=0 . . . 185
12.8.3.18 delete_type_builder(class DynamicTypeBuilder *dtb)=0 . . . 185
12.8.3.19 get_instance() . . . 185
12.8.3.20 get_primitive_type(const TypeKind kind)=0 . . . 185
12.9 DDS::DynamicTypeMember Class Reference . . . 186
12.9.1 Detailed Description . . . 186
12.9.2 Constructor & Destructor Documentation . . . 187
12.9.2.1 DynamicTypeMember() . . . 187
12.9.3 Member Function Documentation . . . 187
12.9.3.1 equals(const class DynamicTypeMember *other)=0 . . . 187
12.9.3.2 get_annotation(class AnnotationDescriptor *descriptor, const unsigned int idx)=0 . . . 187
12.9.3.3 get_annotation_count()=0 . . . 187
12.9.3.4 get_descriptor(class MemberDescriptor *descriptor)=0 . . . 187
12.9.3.5 get_id()=0 . . . 188
12.9.3.6 get_name()=0 . . . 188
12.10 DDS::DynamicTypeSupport Class Reference . . . 188
12.10.1 Detailed Description . . . 188
12.10.2 Constructor & Destructor Documentation . . . 189

12.10.2.1 DynamicTypeSupport() . . . 189
12.10.3 Member Function Documentation . . . 189
12.10.3.1 create_type_support(DynamicType ∗type) . . . 189
12.10.3.2 delete_type_support(DynamicTypeSupport ∗type_support) . . . 189
12.10.3.3 get_type()=0 . . . 189
12.10.3.4 get_type_name()=0 . . . 189
12.11 DDS::MemberDescriptor Class Reference . . . 190
12.11.1 Detailed Description . . . 190
12.11.2 Constructor & Destructor Documentation . . . 190
12.11.2.1 MemberDescriptor() . . . 190
12.11.3 Member Function Documentation . . . 190
12.11.3.1 copy_from(const class MemberDescriptor ∗descriptor) . . . 190
12.11.3.2 equals(const class MemberDescriptor ∗descriptor) . . . 190
12.11.3.3 is_consistent() . . . 191
12.12 DDS::TypeDescriptor Class Reference . . . 191
12.12.1 Detailed Description . . . 191
12.12.2 Constructor & Destructor Documentation . . . 192
12.12.2.1 TypeDescriptor() . . . 192
12.12.3 Member Function Documentation . . . 192
12.12.3.1 copy_from(const class TypeDescriptor ∗other) . . . 192
12.12.3.2 equals(const class TypeDescriptor ∗other) . . . 192
12.12.3.3 is_consistent() . . . 192
12.12.4 Member Data Documentation . . . 192
12.12.4.1 base_type . . . 192
12.12.4.2 bound . . . 193
12.12.4.3 element_type . . . 193
12.12.4.4 key_element_type . . . 193

13 CoreDX DDS Transports **195**
13.1 CoreDX_LmtTransport Struct Reference . . . 195
13.1.1 Detailed Description . . . 195
13.1.2 Friends And Related Function Documentation . . . 195
13.1.2.1 CoreDX_LmtTransport_clear_config(CoreDX_LmtTransportConfig ∗config) . . . 195
13.1.2.2 CoreDX_LmtTransport_create_transport(CoreDX_LmtTransportConfig ∗config) . . . 196
13.1.2.3 CoreDX_LmtTransport_get_default_config(CoreDX_LmtTransportConfig ∗config) . . . 196
13.1.2.4 CoreDX_LmtTransport_get_env_config(CoreDX_LmtTransportConfig ∗config) . . . 196
13.2 CoreDX_LmtTransportConfig Struct Reference . . . 196

13.2.1 Detailed Description . . . 196
13.2.2 Member Data Documentation . . . 196
13.2.2.1 debug_flags . . . 196
13.2.2.2 max_rx_buf_size . . . 197
13.2.2.3 max_tx_size . . . 197
13.2.2.4 so_rcvbuf . . . 197
13.2.2.5 so_sndbuf . . . 197
13.3 CoreDX_TcpTransport Struct Reference . . . 197
13.3.1 Detailed Description . . . 197
13.3.2 Friends And Related Function Documentation . . . 197
13.3.2.1 CoreDX_TcpTransport_clear_config(CoreDX_TcpTransportConfig ∗config) . . . 197
13.3.2.2 CoreDX_TcpTransport_create_transport(CoreDX_TcpTransportConfig ∗config) . . . 198
13.3.2.3 CoreDX_TcpTransport_get_default_config(CoreDX_TcpTransportConfig ∗config) . . . 198
13.3.2.4 CoreDX_TcpTransport_get_env_config(CoreDX_TcpTransportConfig ∗config) . . . 198
13.4 CoreDX_TcpTransportConfig Struct Reference . . . 198
13.4.1 Detailed Description . . . 198
13.4.2 Member Data Documentation . . . 198
13.4.2.1 debug_flags . . . 198
13.4.2.2 dynamic_interfaces . . . 199
13.4.2.3 interfaces . . . 199
13.4.2.4 participant_index . . . 199
13.4.2.5 tx_max_packet_size . . . 199
13.5 CoreDX_UdpTransport Struct Reference . . . 199
13.5.1 Detailed Description . . . 199
13.5.2 Friends And Related Function Documentation . . . 199
13.5.2.1 CoreDX_UdpTransport_clear_config(CoreDX_UdpTransportConfig ∗config) . . . 199
13.5.2.2 CoreDX_UdpTransport_create_transport(CoreDX_UdpTransportConfig ∗config) . . . 200
13.5.2.3 CoreDX_UdpTransport_get_default_config(CoreDX_UdpTransportConfig ∗config) . . . 200
13.5.2.4 CoreDX_UdpTransport_get_env_config(CoreDX_UdpTransportConfig ∗config) . . . 200
13.6 CoreDX_UdpTransportConfig Struct Reference . . . 200
13.6.1 Detailed Description . . . 201
13.6.2 Member Data Documentation . . . 201
13.6.2.1 advertise_meta_multicast . . . 201
13.6.2.2 advertise_user_multicast . . . 201
13.6.2.3 broadcast_address . . . 201
13.6.2.4 debug_flags . . . 201
13.6.2.5 do_meta_broadcast . . . 201

13.6.2.6 dynamic_interfaces . . . 201
13.6.2.7 interfaces . . . 201
13.6.2.8 meta_multicast_address_v4 . . . 201
13.6.2.9 meta_multicast_address_v6 . . . 202
13.6.2.10 multicast_ttl . . . 202
13.6.2.11 participant_index . . . 202
13.6.2.12 rx_init_buffer_size . . . 202
13.6.2.13 rx_max_buffer_size . . . 202
13.6.2.14 rx_meta_multicast . . . 202
13.6.2.15 rx_user_multicast . . . 202
13.6.2.16 so_rcvbuf . . . 202
13.6.2.17 so_sndbuf . . . 202
13.6.2.18 tx_max_packet_size . . . 202
13.6.2.19 tx_meta_multicast . . . 202
13.6.2.20 tx_meta_unicast . . . 203
13.6.2.21 use_ipv4 . . . 203
13.6.2.22 use_ipv6 . . . 203
13.6.2.23 user_multicast_address_v4 . . . 203
13.6.2.24 user_multicast_address_v6 . . . 203

14 CoreDX DDS RPC over DDS API 205

14.1 DDS::rpc::ClientEndpoint Class Reference . . . 205
14.1.1 Detailed Description . . . 205
14.2 DDS::rpc::ClientParams Class Reference . . . 206
14.2.1 Detailed Description . . . 207
14.3 DDS::rpc::ListenerBase Class Reference . . . 207
14.3.1 Detailed Description . . . 207
14.4 DDS::rpc::Replier< TReq, TRep > Class Template Reference . . . 207
14.4.1 Detailed Description . . . 208
14.4.2 Constructor & Destructor Documentation . . . 209
14.4.2.1 Replier() . . . 209
14.4.3 Member Function Documentation . . . 209
14.4.3.1 read_request(Sample< TReq > &request) . . . 209
14.4.3.2 read_request(SampleRef< TReq > request) . . . 209
14.4.3.3 read_requests(int max_samples) . . . 209
14.4.3.4 receive_nondata_samples(bool enable) . . . 209
14.4.3.5 receive_request(Sample< TReq > &request, const DDS::Duration_t &max_wait) . . . 210

14.4.3.6 receive_request(SampleRef< TReq > request, const DDS::Duration_t &max_wait) . . 210
14.4.3.7 receive_requests(const DDS::Duration_t &max_wait) 210
14.4.3.8 receive_requests(uint32_t min_request_count, uint32_t max_request_count, const DDS::Duration_t &max_wait) 210
14.4.3.9 take_request(Sample< TReq > &request) 210
14.4.3.10 take_request(SampleRef< TReq > request) 210
14.4.3.11 take_requests(int max_samples) 211
14.4.3.12 wait_for_requests(const DDS::Duration_t &max_wait) 211
14.4.3.13 wait_for_requests(uint32_t min_count, const DDS::Duration_t &max_wait) 211
14.5 DDS::rpc::ReplierBase Class Reference 211
14.5.1 Detailed Description 211
14.6 DDS::rpc::ReplierListener< TReq, TRep > Class Template Reference 212
14.6.1 Detailed Description 212
14.6.2 Member Function Documentation 212
14.6.2.1 on_request_available(Replier< TReq, TRep > &)=0 212
14.7 DDS::rpc::ReplierParams Class Reference 212
14.7.1 Detailed Description 214
14.8 DDS::rpc::Requester< TReq, TRep > Class Template Reference 214
14.8.1 Detailed Description 216
14.8.2 Constructor & Destructor Documentation 216
14.8.2.1 Requester() 216
14.8.3 Member Function Documentation 216
14.8.3.1 read_replies(uint32_t max_count) 216
14.8.3.2 read_replies(uint32_t max_count, const DDS::SampleIdentity_t &related_request_id) . 216
14.8.3.3 read_replies(const DDS::SampleIdentity_t &related_request_id) 217
14.8.3.4 read_reply(Sample< TRep > &reply) 217
14.8.3.5 read_reply(SampleRef< TRep > reply) 217
14.8.3.6 read_reply(Sample< TRep > &reply, const DDS::SampleIdentity_t &related_request_id) 217
14.8.3.7 read_reply(SampleRef< TRep > reply, const DDS::SampleIdentity_t &related_request_id) 217
14.8.3.8 receive_nondata_samples(bool enable) 217
14.8.3.9 receive_replies(const DDS::Duration_t &max_wait) 217
14.8.3.10 receive_replies(uint32_t min_count, uint32_t max_count, const DDS::Duration_t &max_wait) 218
14.8.3.11 receive_reply(Sample< TRep > &reply, const DDS::Duration_t &timeout) 218
14.8.3.12 receive_reply(SampleRef< TRep > reply, const DDS::Duration_t &timeout) 218
14.8.3.13 receive_reply(Sample< TRep > &reply, const DDS::SampleIdentity_t &relatedRequestId) 218

14.8.3.14 receive_reply(SampleRef< TRep > reply, const DDS::SampleIdentity_t &relatedRequestId) . . . 219
14.8.3.15 take_replies(uint32_t max_count) . . . 219
14.8.3.16 take_replies(uint32_t max_count, const DDS::SampleIdentity_t &related_request_id) . 219
14.8.3.17 take_replies(const DDS::SampleIdentity_t &related_request_id) . . . 219
14.8.3.18 take_reply(Sample< TRep > &reply) . . . 219
14.8.3.19 take_reply(SampleRef< TRep > reply) . . . 220
14.8.3.20 take_reply(Sample< TRep > &reply, const DDS::SampleIdentity_t &related_request_id) 220
14.8.3.21 take_reply(SampleRef< TRep > reply, const DDS::SampleIdentity_t &related_request_id) . . . 220
14.8.3.22 wait_for_replies(uint32_t min_count, const DDS::Duration_t &max_wait) . . . 220
14.8.3.23 wait_for_replies(const DDS::Duration_t &max_wait) . . . 220
14.8.3.24 wait_for_replies(uint32_t min_count, const DDS::Duration_t &max_wait, const DDS::SampleIdentity_t &related_request_id) . . . 221
14.9 DDS::rpc::RequesterListener< TReq, TRep > Class Template Reference . . . 221
14.9.1 Detailed Description . . . 221
14.9.2 Member Function Documentation . . . 222
14.9.2.1 on_reply_available(Requester< TReq, TRep > &)=0 . . . 222
14.10 DDS::rpc::RequesterParams Class Reference . . . 222
14.10.1 Detailed Description . . . 223
14.10.2 Member Function Documentation . . . 223
14.10.2.1 datareader_qos(dds_entity_traits::DataReaderQos qos) . . . 223
14.10.2.2 datawriter_qos(dds_entity_traits::DataWriterQos qos) . . . 223
14.10.2.3 domain_participant(dds_entity_traits::DomainParticipant participant) . . . 223
14.10.2.4 publisher(dds_entity_traits::Publisher publisher) . . . 224
14.10.2.5 reply_topic_name(const std::string &name) . . . 224
14.10.2.6 request_topic_name(const std::string &name) . . . 224
14.10.2.7 requester_listener(RequesterListener< TReq, TRep > *listener) . . . 224
14.10.2.8 service_name(const std::string &name) . . . 224
14.10.2.9 simple_requester_listener(SimpleRequesterListener< TRep > *listener) . . . 224
14.10.2.10 subscriber(dds_entity_traits::Subscriber subscriber) . . . 224
14.11 DDS::rpc::RPCEntity Class Reference . . . 225
14.11.1 Detailed Description . . . 225
14.11.2 Member Function Documentation . . . 225
14.11.2.1 close() . . . 225
14.11.2.2 is_null() const . . . 225
14.12 DDS::rpc::ServiceEndpoint Class Reference . . . 225
14.12.1 Detailed Description . . . 226

14.12.2 Member Function Documentation . . . 226
14.12.2.1 pause() . . . 226
14.12.2.2 resume() . . . 226
14.13 DDS::rpc::ServiceParams Class Reference . . . 227
14.13.1 Detailed Description . . . 228
14.14 DDS::rpc::ServiceProxy Class Reference . . . 228
14.14.1 Detailed Description . . . 229
14.14.2 Member Function Documentation . . . 229
14.14.2.1 bind(const std::string &instance_name) . . . 229
14.14.2.2 get_discoverd_service_instances() const . . . 229
14.14.2.3 get_discovered_service_count() const . . . 229
14.14.2.4 get_discovered_service_instances() const . . . 229
14.14.2.5 unbind() . . . 229
14.14.2.6 wait_for_service(const DDS::Duration_t &maxWait) . . . 230
14.14.2.7 wait_for_service(const DDS::Duration_t &maxWait, std::string instanceName) . . . 230
14.14.2.8 wait_for_services(const DDS::Duration_t &maxWait, uint32_t count) . . . 230
14.14.2.9 wait_for_services(const DDS::Duration_t &maxWait, const std::vector< std::string > &instanceNames) . . . 230
14.15 DDS::rpc::SimpleReplierListener< TReq, TRep > Class Template Reference . . . 230
14.15.1 Detailed Description . . . 231
14.15.2 Member Function Documentation . . . 231
14.15.2.1 process_request(const Sample< TReq > &, const DDS::SampleIdentity_t &)=0 . . . 231
14.16 DDS::rpc::SimpleRequesterListener< TRep > Class Template Reference . . . 231
14.16.1 Detailed Description . . . 231
14.16.2 Member Function Documentation . . . 232
14.16.2.1 process_reply(const Sample< TRep > &, const DDS::SampleIdentity_t &)=0 . . . 232

15 CoreDX DDS RPC over DDS Example 233

15.1 DDS::rpc::example::Foo Struct Reference . . . 233
15.1.1 Detailed Description . . . 233
15.1.2 Constructor & Destructor Documentation . . . 233
15.1.2.1 Foo() . . . 233
15.1.3 Member Function Documentation . . . 234
15.1.3.1 op1(const int param)=0 . . . 234
15.1.3.2 op2()=0 . . . 234
15.2 DDS::rpc::example::Foo_Call Class Reference . . . 234
15.2.1 Detailed Description . . . 234
15.2.2 Member Function Documentation . . . 234

15.2.2.1 op1() const . . . 234
15.2.2.2 op1(const struct Foo_op1_In &_v) . . . 234
15.2.2.3 op2() const . . . 234
15.2.2.4 op2(const struct Foo_op2_In &_v) . . . 234
15.3 DDS::rpc::example::Foo_op1_In Struct Reference . . . 235
15.3.1 Detailed Description . . . 235
15.3.2 Constructor & Destructor Documentation . . . 235
15.3.2.1 Foo_op1_In() . . . 235
15.3.3 Member Data Documentation . . . 235
15.3.3.1 param . . . 235
15.4 DDS::rpc::example::Foo_op1_Out Struct Reference . . . 235
15.4.1 Detailed Description . . . 235
15.4.2 Constructor & Destructor Documentation . . . 236
15.4.2.1 Foo_op1_Out() . . . 236
15.4.3 Member Data Documentation . . . 236
15.4.3.1 return_ . . . 236
15.5 DDS::rpc::example::Foo_op1_Result Class Reference . . . 236
15.5.1 Detailed Description . . . 236
15.5.2 Member Function Documentation . . . 236
15.5.2.1 result() const . . . 236
15.5.2.2 result(const struct Foo_op1_Out &_v) . . . 236
15.5.2.3 unknownEx() const . . . 236
15.5.2.4 unknownEx(DDS::rpc::UnknownException _v) . . . 236
15.6 DDS::rpc::example::Foo_op2_In Struct Reference . . . 237
15.6.1 Detailed Description . . . 237
15.6.2 Constructor & Destructor Documentation . . . 237
15.6.2.1 Foo_op2_In() . . . 237
15.6.3 Member Data Documentation . . . 237
15.6.3.1 dummy . . . 237
15.7 DDS::rpc::example::Foo_op2_Out Struct Reference . . . 237
15.7.1 Detailed Description . . . 237
15.7.2 Constructor & Destructor Documentation . . . 238
15.7.2.1 Foo_op2_Out() . . . 238
15.7.3 Member Data Documentation . . . 238
15.7.3.1 return_ . . . 238
15.8 DDS::rpc::example::Foo_op2_Result Class Reference . . . 238
15.8.1 Detailed Description . . . 238

15.8.2 Member Function Documentation . . . 238
15.8.2.1 result() const . . . 238
15.8.2.2 result(const struct Foo_op2_Out &_v) . . . 238
15.8.2.3 unknownEx() const . . . 238
15.8.2.4 unknownEx(DDS::rpc::UnknownException _v) . . . 238
15.9 DDS::rpc::example::Foo_Reply Struct Reference . . . 239
15.9.1 Detailed Description . . . 239
15.9.2 Constructor & Destructor Documentation . . . 239
15.9.2.1 Foo_Reply() . . . 239
15.9.3 Member Function Documentation . . . 239
15.9.3.1 related_id() . . . 239
15.9.4 Member Data Documentation . . . 239
15.9.4.1 data . . . 239
15.9.4.2 header . . . 239
15.10 DDS::rpc::example::Foo_Request Struct Reference . . . 239
15.10.1 Detailed Description . . . 240
15.10.2 Constructor & Destructor Documentation . . . 240
15.10.2.1 Foo_Request() . . . 240
15.10.3 Member Function Documentation . . . 240
15.10.3.1 request_id() . . . 240
15.10.4 Member Data Documentation . . . 240
15.10.4.1 data . . . 240
15.10.4.2 header . . . 240
15.11 DDS::rpc::example::Foo_Return Class Reference . . . 240
15.11.1 Detailed Description . . . 240
15.11.2 Member Function Documentation . . . 241
15.11.2.1 op1() const . . . 241
15.11.2.2 op1() . . . 241
15.11.2.3 op2() const . . . 241
15.11.2.4 op2(const class Foo_op2_Result &_v) . . . 241
15.11.2.5 unknownOp() const . . . 241
15.11.2.6 unknownOp(DDS::rpc::UnknownOperation _v) . . . 241
15.12 DDS::rpc::example::FooClient Struct Reference . . . 241
15.12.1 Detailed Description . . . 242
15.12.2 Constructor & Destructor Documentation . . . 242
15.12.2.1 FooClient(DDS::rpc::ClientParams &params) . . . 242
15.12.3 Member Function Documentation . . . 242

15.12.3.1 op1(const int param) . . . 242
15.12.3.2 op2() . . . 242
15.13 DDS::rpc::example::FooService Struct Reference . . . 242
15.13.1 Detailed Description . . . 243
15.13.2 Constructor & Destructor Documentation . . . 243
15.13.2.1 FooService(DDS::rpc::ServiceParams &params) . . . 243
15.13.3 Member Function Documentation . . . 243
15.13.3.1 op1(const int param)=0 . . . 243
15.13.3.2 op2()=0 . . . 243

16 CoreDX DDS DynamicTypes **245**
16.1 CDX::ArrayDynamicType Class Reference . . . 245
16.1.1 Detailed Description . . . 246
16.1.2 Member Function Documentation . . . 246
16.1.2.1 get_element(uint32_t n) . . . 246
16.1.2.2 get_element_type() . . . 246
16.1.2.3 get_length() . . . 246
16.1.2.4 set_element(uint32_t n, DynamicType ∗e) . . . 246
16.1.2.5 set_element_type(DynamicType ∗e) . . . 247
16.1.2.6 set_length(uint32_t n) . . . 247
16.1.2.7 set_max_length(uint32_t n) . . . 247
16.2 CDX::BooleanDynamicType Class Reference . . . 248
16.2.1 Detailed Description . . . 248
16.2.2 Member Function Documentation . . . 248
16.2.2.1 get_boolean() . . . 248
16.2.2.2 set_boolean(unsigned char c) . . . 248
16.3 CDX_XmlApi Struct Reference . . . 248
16.3.1 Detailed Description . . . 249
16.4 CDX::CharDynamicType Class Reference . . . 249
16.4.1 Detailed Description . . . 249
16.4.2 Member Function Documentation . . . 249
16.4.2.1 get_char() . . . 249
16.4.2.2 set_char(char c) . . . 249
16.5 CDX::DoubleDynamicType Class Reference . . . 250
16.5.1 Detailed Description . . . 250
16.5.2 Member Function Documentation . . . 250
16.5.2.1 get_double() . . . 250

16.5.2.2 set_double(double c) . . . 250
16.6 CDX::DynamicType Class Reference . . . 251
16.6.1 Detailed Description . . . 251
16.6.2 Member Function Documentation . . . 252
16.6.2.1 get_type() . . . 252
16.7 CDX::DynamicTypeDataReader Class Reference . . . 252
16.7.1 Detailed Description . . . 252
16.8 CDX::DynamicTypeDataWriter Class Reference . . . 252
16.8.1 Detailed Description . . . 253
16.9 CDX::EnumConstant Class Reference . . . 253
16.9.1 Detailed Description . . . 253
16.9.2 Member Data Documentation . . . 253
16.9.2.1 name . . . 253
16.9.2.2 value . . . 253
16.10 CDX::EnumDynamicType Class Reference . . . 253
16.10.1 Detailed Description . . . 254
16.10.2 Member Function Documentation . . . 254
16.10.2.1 get_constant(int32_t index) . . . 254
16.10.2.2 get_constant_by_name(const char ∗name) . . . 254
16.10.2.3 get_constant_by_value(uint32_t val) . . . 254
16.10.2.4 get_num_constants(void) . . . 254
16.10.2.5 get_value(void) . . . 254
16.10.2.6 set_constant(int32_t n, const char ∗name, uint32_t val) . . . 254
16.10.2.7 set_num_constants(int32_t n) . . . 254
16.10.2.8 set_value(uint32_t c) . . . 254
16.11 CDX::FloatDynamicType Class Reference . . . 255
16.11.1 Detailed Description . . . 255
16.11.2 Member Function Documentation . . . 255
16.11.2.1 get_float() . . . 255
16.11.2.2 set_float(float c) . . . 255
16.12 CDX::LongDynamicType Class Reference . . . 255
16.12.1 Detailed Description . . . 256
16.12.2 Member Function Documentation . . . 256
16.12.2.1 get_long() . . . 256
16.12.2.2 set_long(int32_t c) . . . 256
16.13 CDX::LongLongDynamicType Class Reference . . . 256
16.13.1 Detailed Description . . . 257

16.13.2 Member Function Documentation . . . 257
16.13.2.1 get_longlong() . . . 257
16.13.2.2 set_longlong(int64_t c) . . . 257
16.14CDX::OctetDynamicType Class Reference . . . 257
16.14.1 Detailed Description . . . 258
16.14.2 Member Function Documentation . . . 258
16.14.2.1 get_octet() . . . 258
16.14.2.2 set_octet(unsigned char c) . . . 258
16.15CDX::SequenceDynamicType Class Reference . . . 258
16.15.1 Detailed Description . . . 259
16.15.2 Member Function Documentation . . . 259
16.15.2.1 get_element(uint32_t n) . . . 259
16.15.2.2 get_element_type() . . . 259
16.15.2.3 get_length() . . . 260
16.15.2.4 set_element(uint32_t n, DynamicType ∗e) . . . 260
16.15.2.5 set_element_type(DynamicType ∗e) . . . 260
16.15.2.6 set_length(uint32_t n) . . . 260
16.15.2.7 set_max_length(uint32_t n) . . . 261
16.16CDX::ShortDynamicType Class Reference . . . 261
16.16.1 Detailed Description . . . 261
16.16.2 Member Function Documentation . . . 262
16.16.2.1 get_short() . . . 262
16.16.2.2 set_short(short c) . . . 262
16.17CDX::StringDynamicType Class Reference . . . 262
16.17.1 Detailed Description . . . 263
16.17.2 Member Function Documentation . . . 263
16.17.2.1 get_length() . . . 263
16.17.2.2 get_max_length() . . . 263
16.17.2.3 get_string() . . . 263
16.17.2.4 set_max_length(uint32_t n) . . . 263
16.17.2.5 set_string(const char ∗c) . . . 264
16.18CDX::StructDynamicType Class Reference . . . 264
16.18.1 Detailed Description . . . 264
16.18.2 Member Function Documentation . . . 265
16.18.2.1 get_field(uint32_t n) . . . 265
16.18.2.2 get_field_key(uint32_t n) . . . 265
16.18.2.3 get_field_name(uint32_t n) . . . 265

16.18.2.4 get_num_fields() . . . 265
16.18.2.5 set_field(uint32_t n, const char *field_name, DynamicType *e, unsigned char key) . . 266
16.18.2.6 set_num_fields(uint32_t n) . . . 266
16.19 CDX::ULongDynamicType Class Reference . . . 266
16.19.1 Detailed Description . . . 267
16.19.2 Member Function Documentation . . . 267
16.19.2.1 get_ulong() . . . 267
16.19.2.2 set_ulong(uint32_t c) . . . 267
16.20 CDX::ULongLongDynamicType Class Reference . . . 267
16.20.1 Detailed Description . . . 268
16.20.2 Member Function Documentation . . . 268
16.20.2.1 get_ulonglong() . . . 268
16.20.2.2 set_ulonglong(uint64_t c) . . . 268
16.21 CDX::UnionDynamicType Class Reference . . . 268
16.21.1 Detailed Description . . . 269
16.21.2 Member Function Documentation . . . 269
16.21.2.1 get_default_field() . . . 269
16.21.2.2 get_discriminator() . . . 269
16.21.2.3 get_field(uint32_t n) . . . 270
16.21.2.4 get_field_label(uint32_t field, uint32_t label) . . . 270
16.21.2.5 get_field_name(uint32_t n) . . . 270
16.21.2.6 get_field_num_labels(uint32_t field) . . . 270
16.21.2.7 get_num_fields() . . . 270
16.21.2.8 get_selected_field(void) . . . 271
16.21.2.9 set_default_field(int32_t field) . . . 271
16.21.2.10 set_discriminator(DynamicType *d) . . . 271
16.21.2.11 set_field(uint32_t n, const char *field_name, DynamicType *e) . . . 272
16.21.2.12 set_field_label(uint32_t field, uint32_t label, int32_t val) . . . 272
16.21.2.13 set_field_num_labels(uint32_t field, uint32_t n) . . . 272
16.21.2.14 set_num_fields(uint32_t n) . . . 273
16.22 CDX::UShortDynamicType Class Reference . . . 273
16.22.1 Detailed Description . . . 273
16.22.2 Member Function Documentation . . . 274
16.22.2.1 get_ushort() . . . 274
16.22.2.2 set_ushort(unsigned short c) . . . 274

17.1 Class List . . . 275

18 File Documentation **283**

18.1 cpp/dds.hh File Reference . . . 283

18.1.1 Detailed Description . . . 286

18.2 cpp/dds_builtin.hh File Reference . . . 287

18.3 cpp/dds_dtype.hh File Reference . . . 289

18.3.1 Detailed Description . . . 290

18.3.2 Enumeration Type Documentation . . . 290

18.3.2.1 TypeCodeKind . . . 290

18.4 cpp/dds_seq.hh File Reference . . . 291

18.4.1 Detailed Description . . . 291

18.5 cpp/dds_types.h File Reference . . . 292

18.5.1 Detailed Description . . . 293

18.5.2 Typedef Documentation . . . 293

18.5.2.1 DDS_DomainId_t . . . 293

18.5.2.2 DDS_InstanceHandle_t . . . 293

18.5.2.3 DDS_ReturnCode_t . . . 293

18.5.2.4 DDS_StatusMask . . . 293

18.6 cpp/coredx_lmt_transport.h File Reference . . . 294

18.6.1 Detailed Description . . . 294

18.7 cpp/coredx_tcp_transport.h File Reference . . . 295

18.7.1 Detailed Description . . . 295

18.8 cpp/coredx_udp_transport.h File Reference . . . 296

18.8.1 Detailed Description . . . 296

18.9 cpp/function_call.hh File Reference . . . 297

18.9.1 Detailed Description . . . 297

18.10 cpp/qos_provider.hh File Reference . . . 297

18.10.1 Detailed Description . . . 297

18.11 cpp/request_reply.hh File Reference . . . 298

18.11.1 Detailed Description . . . 298

18.12 cpp/sample.hh File Reference . . . 299

18.12.1 Detailed Description . . . 300

18.13 cpp/xml_api.h File Reference . . . 300

18.13.1 Detailed Description . . . 300

18.14 cpp/xtypes.hh File Reference . . . 300

18.14.1 Detailed Description . . . 301

18.15cpp/xtypes_dtype.hh File Reference . . . 301
18.15.1 Detailed Description . . . 302

19 Not Yet Supported **303**

20 Deprecated List **307**

21 Hierarchical Index **309**
21.1 Class Hierarchy . . . 309

22 Namespace Details **313**
22.1 Namespace List . . . 313
22.2 DDS Namespace Reference . . . 313
22.2.1 Detailed Description . . . 318
22.2.2 Typedef Documentation . . . 318
22.2.2.1 DomainId_t . . . 318
22.2.2.2 GuidPrefix_t . . . 318
22.2.2.3 InstanceHandle_t . . . 318
22.2.2.4 ReturnCode_t . . . 318
22.2.2.5 StatusMask . . . 318
22.2.3 Enumeration Type Documentation . . . 318
22.2.3.1 DestinationOrderQosPolicyKind . . . 318
22.2.3.2 DiscoveryQosPolicyDiscoveryKind . . . 319
22.2.3.3 DurabilityQosPolicyKind . . . 319
22.2.3.4 HistoryQosPolicyKind . . . 319
22.2.3.5 LivelinessQosPolicyKind . . . 319
22.2.3.6 OwnershipQosPolicyKind . . . 320
22.2.3.7 PresentationQosPolicyAccessScopeKind . . . 320
22.2.3.8 ReliabilityQosPolicyKind . . . 320
22.2.3.9 TypeConsistencyKind . . . 320
22.3 DDS::rpc Namespace Reference . . . 320
22.3.1 Detailed Description . . . 321
22.4 DDS::rpc::example Namespace Reference . . . 321
22.4.1 Detailed Description . . . 322

Index **323**

Chapter 1

Introduction

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 the C++ Application Programming Interface (API).

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. CoreDX DDS includes C, C++, Java, and C# language bindings for application integration. This reference manual describes the CoreDX DDS "C++" language binding.

Document Organization

The reference documentation is organized in the following categories:

1. **DDS Entities** (p. 3)
   This includes the primary objects with which an application must interact to enable DDS publish-subscribe communications.
2. **DDS Quality of Service** (p. 4)
   This section documents the Quality of Service structures that configure the behavior of the CoreDX DDS middleware.
3. **DDS Conditions, Listeners, and WaitSets** (p. 8)
   This section covers the various structures and concepts involved in delivering events to the application.
4. **DDS Status Structures** (p. 13)
   This section describes the types of status maintained by the infrastructure.
5. **X-Types Dynamic Types** (p. 19)
   This includes routines and objects to support **dynamic** data types.
6. **CoreDX DDS Transports** (p. 15)
   CoreDX DDS Tranpsorts
7. **CoreDX DDS RPC** (p. 16)
   CoreDX DDS RPC
8. **CoreDX DDS 'Sample' objects.** (p. 12)
   DDS::Sample<T> And Related

Also, a list of API items which are not yet supported by CoreDX DDS are provided **here** (p. 303).

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 Entities

Classes

- class **DDS::Entity**

  Base class for all DDS Entities.
- class **DDS::DomainEntity**

  Base class for all DDS Domain Entities.
- class **DDS::DomainParticipantFactory**

  DomainParticipantFactory *(p. 54) constructs DomainParticipants. The.*
- class **DDS::DomainParticipant**

  The ***DomainParticipant*** *(p. 44) is used to configure, create and destroy* ***Publisher*** *(p. 57),* ***Subscriber*** *(p. 65) and* ***Topic*** *(p. 69) objects.*
- class **DDS::Publisher**

  The ***Publisher*** *(p. 57) configures, creates, manages and destroys DataWriters.*
- class **DDS::Subscriber**

  The ***Subscriber*** *(p. 65) configures, creates, manages and destroys DataReaders.*
- class **DDS::TopicDescription**

  TopicDescription *(p. 72) is an abstract 'class' that provides the foundation for* ***Topic*** *(p. 69),* ***ContentFilteredTopic*** *(p. 24), and MultiTopic.*
- class **DDS::Topic**

  Topic *(p. 69) is the basic description of data to be published or subscribed.*
- class **DDS::ContentFilteredTopic**

  ContentFilteredTopic *(p. 24) provides a topic that may exclude data based on a specified filter. The* ***ContentFilteredTopic*** *(p. 24) 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*** *(p. 26) associated with the* ***ContentFilteredTopic*** *(p. 24).*
- class **DDS::DataWriter**

  The ***DataWriter*** *(p. 37) entity provides an interface for the application to publish (write) data.*
- class **DDS::DataReader**

  The ***DataReader*** *(p. 26) entity allows the application to subscribe to and read data.*

1.1.1 Detailed Description

1.2 DDS Quality of Service

Classes

- class **DDS::DomainParticipantFactoryQos**

  *Structure that holds **DomainParticipantFactory** (p. 54) Quality of Service policies.*
- class **DDS::DomainParticipantQos**

  *Structure that holds **DomainParticipant** (p. 44) Quality of Service policies.*
- class **DDS::TopicQos**

  *Structure that holds **Topic** (p. 69) Quality of Service policies.*
- class **DDS::DataWriterQos**

  *Structure that holds **DataWriter** (p. 37) Quality of Service policies.*
- class **DDS::PublisherQos**

  *Structure that holds **Publisher** (p. 57) Quality of Service policies.*
- class **DDS::DataReaderQos**

  *Structure that holds **DataReader** (p. 26) Quality of Service policies.*
- class **DDS::SubscriberQos**

  *Structure that holds **Subscriber** (p. 65) Quality of Service policies.*
- class **DDS::QosProvider**

  ***QosProvider** (p. 105) loads QoS settings from a library and provides interfaces to access entity specific QoS Policies.*

Typedefs

- typedef short **DDS::DataRepresentationId_t**

  Indicates the form of data, for example CDR or XML.

Functions

- **DDS::QosProvider::QosProvider** (const char ∗uri, const char ∗profile, **CDX_XmlApi** ∗xml_parser)

  *Constructor. Creates a new **QosProvider** (p. 105) instance that loads QoS settings from a library.*
- **DDS::QosProvider::∼QosProvider** ()

  *Destructor. Returns a **QosProvider** (p. 105) back to the factory.*
- DomainParticipantFactoryQos ∗ **DDS::QosProvider::get_participantfactory_qos** (const char ∗id)

  *Access a **DomainParticipantFactory** (p. 54) QoS.*
- void **DDS::QosProvider::return_participantfactory_qos** (DomainParticipantFactoryQos ∗qos)

  Return a DomainParticipanFactoryQos back to the factory.
- DomainParticipantQos ∗ **DDS::QosProvider::get_participant_qos** (const char ∗id)

  *Access a **DomainParticipant** (p. 44) QoS.*
- void **DDS::QosProvider::return_participant_qos** (DomainParticipantQos ∗qos)

  *Return a **DomainParticipantQos** (p. 78) back to the factory.*
- TopicQos ∗ **DDS::QosProvider::get_topic_qos** (const char ∗id)

  *Access a **Topic** (p. 69) QoS.*
- void **DDS::QosProvider::return_topic_qos** (TopicQos ∗qos)

  *Return a **TopicQos** (p. 80) back to the factory.*
- SubscriberQos ∗ **DDS::QosProvider::get_subscriber_qos** (const char ∗id)

  *Access a **Subscriber** (p. 65) QoS.*
- void **DDS::QosProvider::return_subscriber_qos** (SubscriberQos ∗qos)

*Return a **SubscriberQos** (p. 79) back to the factory.*

- PublisherQos * **DDS::QosProvider::get_publisher_qos** (const char *id)

  *Access a **Publisher** (p. 57) QoS.*
- void **DDS::QosProvider::return_publisher_qos** (PublisherQos *qos)

  *Return a **PublisherQos** (p. 78) back to the factory.*
- DataReaderQos * **DDS::QosProvider::get_datareader_qos** (const char *id)

  *Access a **DataReader** (p. 26) QoS.*
- void **DDS::QosProvider::return_datareader_qos** (DataReaderQos *qos)

  *Return a **DataReaderQos** (p. 75) back to the factory.*
- DataWriterQos * **DDS::QosProvider::get_datawriter_qos** (const char *id)

  *Access a **DataWriter** (p. 37) QoS.*
- void **DDS::QosProvider::return_datawriter_qos** (DataWriterQos *qos)

  *Return a **DataWriterQos** (p. 76) back to the factory.*

1.2.1 Detailed Description

1.2.2 Function Documentation

1.2.2.1 DataReaderQos* DDS::QosProvider::get_datareader_qos (const char * *id*)

Access a **DataReader** (p. 26) QoS.

If 'id' is NULL, then the first instance of **DataReaderQos** (p. 75) 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.

1.2.2.2 DataWriterQos* DDS::QosProvider::get_datawriter_qos (const char * *id*)

Access a **DataWriter** (p. 37) QoS.

If 'id' is NULL, then the first instance of **DataWriterQos** (p. 76) 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.

1.2.2.3 DomainParticipantQos* DDS::QosProvider::get_participant_qos (const char * *id*)

Access a **DomainParticipant** (p. 44) QoS.

If 'id' is NULL, then the first instance of **DomainParticipantQos** (p. 78) 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.

1.2.2.4 DomainParticipantFactoryQos* DDS::QosProvider::get_participantfactory_qos (const char * *id*)

Access a **DomainParticipantFactory** (p. 54) QoS.

If 'id' is NULL, then the first instance of **DomainParticipantFactoryQos** (p. 77) 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.

1.2.2.5 PublisherQos* DDS::QosProvider::get_publisher_qos (const char * *id*)

Access a **Publisher** (p. 57) QoS.

If 'id' is NULL, then the first instance of **PublisherQos** (p. 78) 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.

1.2.2.6 SubscriberQos* DDS::QosProvider::get_subscriber_qos (const char * *id*)

Access a **Subscriber** (p. 65) QoS.

If 'id' is NULL, then the first instance of **SubscriberQos** (p. 79) 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.

1.2.2.7 TopicQos* DDS::QosProvider::get_topic_qos (const char * *id*)

Access a **Topic** (p. 69) QoS.

If 'id' is NULL, then the first instance of **TopicQos** (p. 80) 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.

1.2.2.8 DDS::QosProvider::QosProvider (const char * *uri,* const char * *profile,* **CDX_XmlApi** * *xml_parser*)

Constructor. Creates a new **QosProvider** (p. 105) 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** (p. 105) selects QoS policies from the library based on the 'profile' parameter. The 'xml_parser' paramater provides the implementation of an XML parser that should be used to parse the QoS library document. CoreDX DDS provides an implementation of **CDX_XmlApi** (p. 248) based on libxml2 named 'cdx_libxml2_impl' available in the include/dds/xml_api.h header file. Alternate implemenations are also available, including on based on RapidXML.

1.2.2.9 void DDS::QosProvider::return_datareader_qos (**DataReaderQos** * *qos*)

Return a **DataReaderQos** (p. 75) back to the factory.

This will reclaim any resources allocated by the **QosProvider** (p. 105).

1.2.2.10 void DDS::QosProvider::return_datawriter_qos (**DataWriterQos** * *qos*)

Return a **DataWriterQos** (p. 76) back to the factory.

This will reclaim any resources allocated by the **QosProvider** (p. 105).

1.2.2.11 void DDS::QosProvider::return_participant_qos (**DomainParticipantQos** * *qos*)

Return a **DomainParticipantQos** (p. 78) back to the factory.

This will reclaim any resources allocated by the **QosProvider** (p. 105).

1.2.2.12 void DDS::QosProvider::return_participantfactory_qos (**DomainParticipantFactoryQos** * *qos*)

Return a DomainParticipanFactoryQos back to the factory.

This will reclaim any resources allocated by the **QosProvider** (p. 105).

1.2.2.13 void DDS::QosProvider::return_publisher_qos (**PublisherQos** * *qos*)

Return a **PublisherQos** (p. 78) back to the factory.

This will reclaim any resources allocated by the **QosProvider** (p. 105).

1.2.2.14 void DDS::QosProvider::return_subscriber_qos (SubscriberQos * *qos*)

Return a **SubscriberQos** (p. 79) back to the factory.

This will reclaim any resources allocated by the **QosProvider** (p. 105).

1.2.2.15 void DDS::QosProvider::return_topic_qos (TopicQos * *qos*)

Return a **TopicQos** (p. 80) back to the factory.

This will reclaim any resources allocated by the **QosProvider** (p. 105).

1.2.2.16 DDS::QosProvider::~QosProvider ( )

Destructor. Returns a **QosProvider** (p. 105) back to the factory.

This will reclaim any resources allocated by the **QosProvider** (p. 105).

1.3 DDS Conditions, Listeners, and WaitSets

Modules

- **DDS Listeners**
- **DDS Conditions**
- **DDS WaitSets**

1.3.1 Detailed Description

1.4 DDS Listeners

Classes

- class **DDS::DataWriterListener**

  *The **DataWriterListener** (p. 109) provides asynchronous notification of **DataWriter** (p. 37) events.*
- class **DDS::PublisherListener**

  *The **PublisherListener** (p. 110) provides asynchronous notification of **Publisher** (p. 57) events.*
- class **DDS::TopicListener**

  *The **TopicListener** (p. 112) provides asynchronous notification of **Topic** (p. 69) events.*
- class **DDS::DataReaderListener**

  *The **DataReaderListener** (p. 107) provides asynchronous notification of **DataReader** (p. 26) events.*
- class **DDS::SubscriberListener**

  *The **SubscriberListener** (p. 111) provides asynchronous notification of **Subscriber** (p. 65) events.*
- class **DDS::DomainParticipantListener**

  *The **DomainParticipantListener** (p. 110) provides asynchronous notification of **DomainParticipant** (p. 44) events.*

1.4.1 Detailed Description

1.5 DDS Conditions

Classes

- class **DDS::Condition**

  *A **Condition** (p. 23) can be added to a **WaitSet** (p. 73) to provide synchronous event notification.*
- class **DDS::GuardCondition**

  *A **GuardCondition** (p. 56) is a condition where the **trigger_value** is under application control.*
- class **DDS::StatusCondition**

  *A **StatusCondition** (p. 64) is a condition associated with an **Entity** (p. 55).*
- class **DDS::ReadCondition**

  *A **ReadCondition** (p. 63) is a specialized **Condition** (p. 23) associated with a **DataReader** (p. 26).*
- class **DDS::QueryCondition**

  *A **QueryCondition** (p. 61) is a specialized **ReadCondition** (p. 63) which includes a filter.*

1.5.1 Detailed Description

1.6 DDS WaitSets

Classes

- class **DDS::WaitSet**

  *A **WaitSet** (p. 73) maintains a set of **Condition** (p. 23) objects and allows the application to wait until one or more of them have a **trigger_value** of TRUE.*

1.6.1 Detailed Description

1.7 CoreDX DDS 'Sample' objects.

Classes

- class **DDS::Sample**< **T** >

  Sample *(p. 122) holds the data and related meta information.*
- class **DDS::SampleRef**< **T** >

  SampleRef *(p. 125) holds a reference to data and related meta information.*
- class **DDS::SampleIterator**< **T, IsConst** >

  Provides a means to iterate over a collection of Samples.
- class **DDS::LoanedSamples**< **T** >

  Holds a collection of sample data and related meta-data.
- class **DDS::SharedSamples**< **T** >

  Holds a reference to a collection of sample data and related meta-data.
- class **DDS::WriteSample**< **T** >

  Holds sample data intended for transmission.
- class **DDS::WriteSampleRef**< **T** >

  Holds a reference to sample data intended for transmission.

1.7.1 Detailed Description

These classes are templates based on application data types.

Note

The Sample objects are currently used only in the RPC API.

1.8 DDS Status Structures

Classes

- struct **InconsistentTopicStatus**

  Status related to the on_inconsistent_topic listener methods of the DDS_TopicListener structure.
- struct **SampleLostStatus**

  Status related to the on_sample_lost listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.
- struct **SampleRejectedStatus**

  Status related to the on_sample_rejected listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.
- struct **LivelinessLostStatus**

  Status related to the on_liveliness_lost listener methods of the DDS_DataWriter, DDS_Publisher, and DDS_DomainParticipant structures.
- struct **LivelinessChangedStatus**

  Status related to the on_liveliness_changed listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.
- struct **OfferedDeadlineMissedStatus**

  Status related to the on_offered_deadline_missed listener methods of the DDS_DataWriter, DDS_Publisher, and DDS_DomainParticipant structures.
- struct **RequestedDeadlineMissedStatus**

  Status related to the on_requested_deadline_missed listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.
- struct **QosPolicyCount**

  Holds a DDS_QosPolicyId_t value and a counter to indicate the number of events associated with that PolicyId.
- struct **OfferedIncompatibleQosStatus**

  Status related to the on_offered_incompatible_qos listener methods of the DDS_DataWriter, DDS_Publisher, and DDS_DomainParticipant structures.
- struct **RequestedIncompatibleQosStatus**

  Status related to the on_requested_incompatible_qos listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.
- struct **PublicationMatchedStatus**

  Status related to the on_publication_matched listener methods of the DDS_DataWriter, DDS_Publisher, and DDS_DomainParticipant structures.
- struct **SubscriptionMatchedStatus**

  Status related to the on_subscription_matched listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.

1.8.1 Detailed Description

- InconsistentTopicStatus
- LivelinessChangedStatus
- LivelinessLostStatus
- OfferedDeadlineMissedStatus
- OfferedIncompatibleQosStatus
- PublicationMatchedStatus

- RequestedDeadlineMissedStatus
- RequestedIncompatibleQosStatus
- SampleLostStatus
- SampleRejectedStatus
- SubscriptionMatchedStatus

1.9 CoreDX DDS Transports

Classes

- struct **CoreDX_UdpTransportConfig**

  Structure that holds UDP Transport configuration items.
- struct **CoreDX_SslTransportConfig**

  Structure that holds SSL Transport configuration items.
- struct **CoreDX_TcpTransportConfig**

  Structure that holds TCP Transport configuration items.
- struct **CoreDX_LmtTransportConfig**

  Structure that holds LMT Transport configuration items.
- struct **CoreDX_UdpTransport**

  An instance of a Transport that can be added to a DomainParticipant.
- struct **CoreDX_SslTransport**

  An instance of a Transport that can be added to a DomainParticipant.
- struct **CoreDX_TcpTransport**

  An instance of a Transport that can be added to a DomainParticipant.
- struct **CoreDX_LmtTransport**

  An instance of a Transport that can be added to a DomainParticipant.

1.9.1 Detailed Description

1.10 CoreDX DDS RPC

Files

- file **example_foo_service.hh**

  Example data types generated for an RPC Service.

Classes

- class **DDS::rpc::ListenerBase**

  Basis for all rpc related listener classes.
- class **DDS::rpc::SimpleReplierListener< TReq, TRep >**

  SimpleReplierListener *(p. 230) can be installed on a* ***Replier*** *(p. 207).*
- class **DDS::rpc::ReplierListener< TReq, TRep >**

  ReplierLister can be installed on a ***Replier*** *(p. 207).*
- class **DDS::rpc::SimpleRequesterListener< TRep >**

  SimpleReqeusterListener can be installed on a ***Requester*** *(p. 214).*
- class **DDS::rpc::RequesterListener< TReq, TRep >**

  RequesterListener *(p. 221) can be installed on a* ***Requester*** *(p. 214).*
- class **DDS::rpc::RPCEntity**

  RPCEntity *(p. 225) is a base abstract class.*
- class **DDS::rpc::ServiceProxy**

  ServiceProxy *(p. 228).*
- class **DDS::rpc::Requester< TReq, TRep >**

  A ***Requester*** *(p. 214) sends requests and receives replies.*
- class **DDS::rpc::ReplierBase**

  Provides a generic basis for all ***Replier*** *(p. 207) objects.*
- class **DDS::rpc::Replier< TReq, TRep >**

  A replier receives requests and send replies.
- class **DDS::rpc::RequesterParams**

  Used to pass configuration parameters when constructing a ***Requester*** *(p. 214).*
- class **DDS::rpc::ReplierParams**

  Used to pass configuration parameters when constructing a ***Replier*** *(p. 207).*
- class **DDS::rpc::ServiceParams**

  Used to pass configuration parameters when constructing a Service.
- class **DDS::rpc::ClientParams**

  Used to pass configuration parameters when constructing a Client.
- class **DDS::rpc::ServiceEndpoint**
- class **DDS::rpc::ClientEndpoint**
- struct **DDS::rpc::example::Foo_op1_In**
- struct **DDS::rpc::example::Foo_op1_Out**
- class **DDS::rpc::example::Foo_op1_Result**
- struct **DDS::rpc::example::Foo_op2_In**
- struct **DDS::rpc::example::Foo_op2_Out**
- class **DDS::rpc::example::Foo_op2_Result**
- class **DDS::rpc::example::Foo_Call**
- struct **DDS::rpc::example::Foo_Request**
- class **DDS::rpc::example::Foo_Return**

- struct **DDS::rpc::example::Foo_Reply**
- struct **DDS::rpc::example::Foo**
- struct **DDS::rpc::example::FooClient**
- struct **DDS::rpc::example::FooService**

Enumerations

1.10.1 Detailed Description

1.10.2 Enumeration Type Documentation

1.10.2.1 enum DDS::rpc::ServiceStatus

Indicates the status of a **ServiceEndpoint** (p. 225)

Enumerator

CLOSED The **ServiceEndpoint** (p. 225) has been closed.

PAUSED The **ServiceEndpoint** (p. 225) is paused.

RUNNING The **ServiceEndpoint** (p. 225) is running.

1.11 CoreDX DDS XML API

Classes

- struct **CDX_XmlApi**

  Provides a generic XML Parser capability.

Variables

- **CDX_XmlApi cdx_libxml2_impl**

  the "dds_libxml2api" library provides this implementation that is based on libxml2
- **CDX_XmlApi cdx_xml_impl**

  the "dds_xml" library provides this internal XML parser implementation

1.11.1 Detailed Description

1.12 X-Types Dynamic Types

Files

- file **xtypes.hh**

  Provides the X-Types DDS_DynamicDataReader and DDS_DynamicDataWriter API.
- file **xtypes_dtype.hh**

  Provides the X-Types DDS_DynamicType, DDS_DynamicData, and related APIs.

Classes

- class **DDS::DynamicDataReader**

  *Provides a **DataReader** (p. 26) interface that is tailored to support reading an X-Types defined **DynamicData** (p. 151) data type. The specific **DynamicType** (p. 170) must have been registered previously with the **DomainParticipant** (p. 44).*
- class **DDS::DynamicDataWriter**

  *A **DataWriter** (p. 37) that writes '**DynamicData** (p. 151)' data elements. Data written by this **DataWriter** (p. 37) can be read by a standard **DataReader** (p. 26) or by a **DynamicDataReader** (p. 169), if the QoS and data types match.*
- class **DDS::DynamicTypeBuilderFactory**

  *An instance of this type is responsible for creating **DynamicType** (p. 170) and **DynamicTypeSupport** (p. 188) objects.*
- class **DDS::DynamicTypeSupport**

  *The **DynamicTypeSupport** (p. 188) interface extends the TypeSupport interface defined by the **DDS** (p. 313) specification. This TypeSupport operates on **DynamicData** (p. 151) samples.*
- class **DDS::AnnotationDescriptor**

  *A **AnnotationDescriptor** (p. 149) object comprises the state of an annotation as it is applied to some element.*
- class **DDS::TypeDescriptor**

  *A **TypeDescriptor** (p. 191) comprises the state of a type.*
- class **DDS::MemberDescriptor**

  *A **MemberDescriptor** (p. 190) comprises the state of a **DynamicTypeMember** (p. 186).*
- class **DDS::DynamicTypeMember**

  *A **DynamicTypeMember** (p. 186) represents a "member" of a type. A "member" in this sense may be a member of an aggregated type, a constant within an enumeration, or some other type substructure.*
- class **DDS::DynamicTypeBuilder**

  *A **DynamicTypeBuilder** (p. 173) object represents the state of a particular type defined according to the Type System. It is used to instantiate concrete **DynamicType** (p. 170) objects.*
- class **DDS::DynamicType**

  *An instance of **DynamicType** (p. 170) represent a type's schema: its physical name, kind, member definitions (if any), and so on.*
- class **DDS::DynamicDataFactory**

  *This type is responsible for creating **DynamicData** (p. 151) instances.*
- class **DDS::DynamicData**

  *A **DynamicData** (p. 151) object represents an individual data sample. It provides reflective getters and setters for the members of that sample.*

Functions

- DDS::TypeObject * **DDS::TypecodeQosPolicy_to_typeobject** (TypecodeQosPolicy *tc_qos)

  Helper function to get DDS::TypeObject from a discovered (or otherwise obtained) TypeCode byte buffer.

1.12.1 Detailed Description

1.13 CoreDX DDS DynamicTypes

Classes

- class **CDX::DynamicType**

  DynamicType *(p. 251) is an object that enhances CoreDX DDS with the facilities to process dynamic data types (in other words, data types that are not defined at compile time).*
- class **CDX::OctetDynamicType**

  Contains an octet.
- class **CDX::BooleanDynamicType**

  Contains a boolean.
- class **CDX::CharDynamicType**

  Contains a char.
- class **CDX::ShortDynamicType**

  Contains a short.
- class **CDX::UShortDynamicType**

  Contains an unsigned short.
- class **CDX::LongDynamicType**

  Contains a long.
- class **CDX::ULongDynamicType**

  Contains an unsigned long.
- class **CDX::LongLongDynamicType**

  Contains a long long.
- class **CDX::ULongLongDynamicType**

  Contains an unsigned long long.
- class **CDX::FloatDynamicType**

  Contains a float.
- class **CDX::DoubleDynamicType**

  Contains a double.
- class **CDX::StringDynamicType**

  Contains a string.
- class **CDX::ArrayDynamicType**

  Contains an array.
- class **CDX::SequenceDynamicType**

  Contains a sequence.
- class **CDX::EnumConstant**

  Represents an enum constant value.
- class **CDX::EnumDynamicType**

  Represents an enum (set of constants)
- class **CDX::StructDynamicType**

  Represents a struct of members. Each member has an associated data type.
- class **CDX::UnionDynamicType**

  Represents a union.
- class **CDX::DynamicTypeDataReader**

  *A DataReader that reads '**DynamicType** (p. 251)' data elements A **DynamicTypeDataReader** (p. 252) can read data published by a normal DataWriter or by a **DynamicTypeDataWriter** (p. 252), if the QoS and data types match.*
- class **CDX::DynamicTypeDataWriter**

  *A DataWriter that writes '**DynamicType** (p. 251)' data elements. Data written by this DataWriter can be read by a standard DataReader or by a **DynamicTypeDataReader** (p. 252), if the QoS and data types match.*

1.13.1 Detailed Description

Chapter 2

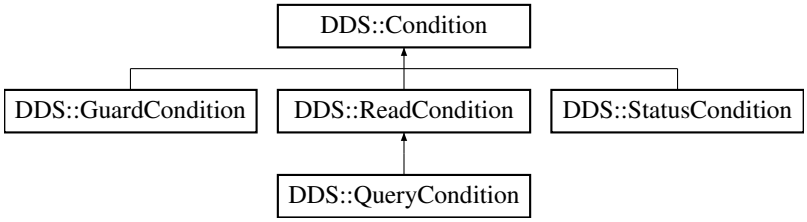
Primary DDS Entities and Types

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

2.1 DDS::Condition Class Reference

A **Condition** (p. 23) can be added to a **WaitSet** (p. 73) to provide synchronous event notification.

Inheritance diagram for DDS::Condition:



Public Member Functions

- virtual bool **get_trigger_value** ()

  *This routine returns the current value of the **trigger_value** in **Condition** (p. 23) **c**.*

2.1.1 Detailed Description

A **Condition** (p. 23) can be added to a **WaitSet** (p. 73) to provide synchronous event notification.

A **Condition** (p. 23) has a **trigger_value** which can be TRUE or FALSE.

2.1.2 Member Function Documentation

2.1.2.1 virtual bool DDS::Condition::get_trigger_value ( ) [virtual]

This routine returns the current value of the **trigger_value** in **Condition** (p. 23) **c**.

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

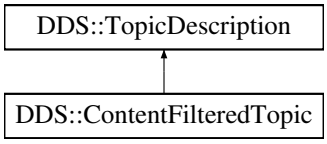
A zero return value indicates that the **trigger_value** is FALSE.

Reimplemented in **DDS::ReadCondition** (p. 64), **DDS::StatusCondition** (p. 65), and **DDS::GuardCondition** (p. 57).

2.2 DDS::ContentFilteredTopic Class Reference

ContentFilteredTopic (p. 24) provides a topic that may exclude data based on a specified filter. The **ContentFilteredTopic** (p. 24) 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** (p. 26) associated with the **ContentFilteredTopic** (p. 24).

Inheritance diagram for DDS::ContentFilteredTopic:



Public Member Functions

- const char ∗ **get_type_name** ()

  *This operation returns **type_name** of the provided **TopicDescription** (p. 72).*
- const char ∗ **get_name** ()

  *This operation returns **topic name** of the provided **TopicDescription** (p. 72).*
- **Topic** ∗ **get_related_topic** ()

  *This returns the real **Topic** (p. 69) associated with the **ContentFilteredTopic** (p. 24).*
- **ReturnCode_t get_expression_parameters** (**StringSeq** &params)

  *This accesses the current set of parameters used by the **ContentFilteredTopic** (p. 24).*
- **ReturnCode_t set_expression_parameters** (const **StringSeq** &params)

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

2.2.1 Detailed Description

ContentFilteredTopic (p. 24) provides a topic that may exclude data based on a specified filter. The **ContentFilteredTopic** (p. 24) 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** (p. 26) associated with the **ContentFilteredTopic** (p. 24).

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

DomainParticipant::create_contentfilteredtopic() (p. 47)

2.2.2 Member Function Documentation

2.2.2.1 ReturnCode_t DDS::ContentFilteredTopic::get_expression_parameters (StringSeq & *params*)

This accesses the current set of parameters used by the **ContentFilteredTopic** (p. 24).

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

2.2.2.2 const char* DDS::ContentFilteredTopic::get_name ( ) [virtual]

This operation returns **topic name** of the provided **TopicDescription** (p. 72).

The returned string is owned by the **TopicDescription (p. 72)**, do not free it.

Reimplemented from **DDS::TopicDescription** (p. 72).

2.2.2.3 Topic* DDS::ContentFilteredTopic::get_related_topic ( )

This returns the real **Topic** (p. 69) associated with the **ContentFilteredTopic** (p. 24).

That is, the **Topic** (p. 69) provided when the **ContentFilteredTopic** (p. 24) was created.

2.2.2.4 const char* DDS::ContentFilteredTopic::get_type_name ( ) [virtual]

This operation returns **type_name** of the provided **TopicDescription** (p. 72).

The returned string is owned by the **TopicDescription (p. 72)**, do not free it.

Reimplemented from **DDS::TopicDescription** (p. 73).

2.2.2.5 ReturnCode_t DDS::ContentFilteredTopic::set_expression_parameters (const StringSeq & *params*)

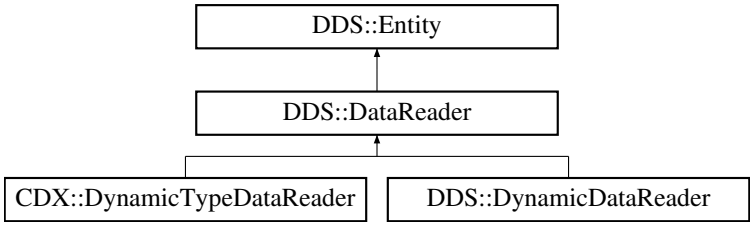
This specifies a new set of parameters for use with the filter_expression.

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 DDS::DataReader Class Reference

The **DataReader** (p. 26) entity allows the application to subscribe to and read data.

Inheritance diagram for DDS::DataReader:



Public Member Functions

- **InstanceHandle_t get_instance_handle** ()

  *This operation returns the InstanceHandle_t that identifies the **DataReader** (p. 26).*
- **StatusMask get_status_changes** ()

  *This returns the list of **triggered** communication statuses in the **DataReader** (p. 26).*
- **ReturnCode_t enable** ()

  *Enables the **DataReader** (p. 26).*
- **StatusCondition** ∗ **get_statuscondition** ()

  *Gets the **StatusCondition** (p. 64) associated with this **Entity** (p. 55). NOTE: The entity maintains ownership of the statuscondition, and it remains valid only as long as the **Entity** (p. 55) is valid. Destroying the **Entity** (p. 55) also destroys the status condition.*
- **ReadCondition** ∗ **create_readcondition** (SampleStateMask sample_states, ViewStateMask view_states, InstanceStateMask instance_states)

  *Creates a **ReadCondition** (p. 63) that is associated with this **DataReader** (p. 26).*
- **ReturnCode_t delete_readcondition** (**ReadCondition** ∗a_condition)

  *Destroys a **ReadCondition** (p. 63) (or **QueryCondition** (p. 61)).*
- **ReturnCode_t delete_contained_entities** ()

  *This operation deletes all the **ReadCondition** (p. 63) and **QueryCondition** (p. 61) objects previously created by means of the DataReader_create_readcondition() and DataReader_create_querycondition() operations.*
- **ReturnCode_t set_qos** (const **DataReaderQos** &qos)

  *Sets the **DataReaderQos** (p. 75) values.*
- **ReturnCode_t get_qos** (**DataReaderQos** &qos)

  *Returns the current **DataReaderQos** (p. 75) settings held in the **DataReader** (p. 26) **dr**.*
- **ReturnCode_t set_listener** (**DataReaderListener** ∗a_listener, **StatusMask** mask)

  *Installs a **DataReaderListener** (p. 107) on **DataReader** (p. 26) **dr**.*
- **DataReaderListener** ∗ **get_listener** ()

  *This operation returns the currently installed **DataReaderListener** (p. 107).*
- **TopicDescription** ∗ **get_topicdescription** ()

  *Returns the **TopicDescription** (p. 72) associated with **DataReader** (p. 26) **dr**.*
- **Subscriber** ∗ **get_subscriber** ()

  *Returns the Subscribier that contains **DataReader** (p. 26) **dr**.*
- **ReturnCode_t get_sample_rejected_status** (SampleRejectedStatus ∗status)

  *Provides access to the current SampleRejectedStatus of the **DataReader** (p. 26).*

- **ReturnCode_t get_liveliness_changed_status** (LivelinessChangedStatus ∗status)

  *Provides access to the current LivelinessChangedStatus of the **DataReader** (p. 26).*
- **ReturnCode_t get_requested_deadline_missed_status** (RequestedDeadlineMissedStatus ∗status)

  *Provides access to the current RequestedDeadlineMissedStatus of the **DataReader** (p. 26).*
- **ReturnCode_t get_requested_incompatible_qos_status** (RequestedIncompatibleQosStatus ∗status)

  *Provides access to the current RequestedIncompatibleQosStatus of the **DataReader** (p. 26).*
- **ReturnCode_t get_subscription_matched_status** (SubscriptionMatchedStatus ∗status)

  *Provides access to the current SubscriptionMatchedStatus of the **DataReader** (p. 26).*
- **ReturnCode_t get_sample_lost_status** (SampleLostStatus ∗status)

  *Provides access to the current SampleLostStatus of the **DataReader** (p. 26).*
- **ReturnCode_t wait_for_historical_data** (const **Duration_t** &max_wait)

  This routine blocks until all 'historical' data is received.
- **ReturnCode_t get_matched_publications** (**InstanceHandleSeq** ∗publication_handles)

  *This operation retrieves the list of DataWriters currently matched with the **DataReader** (p. 26) **dr**.*
- **ReturnCode_t get_matched_publication_data** (**PublicationBuiltinTopicData** ∗publication_data, const **InstanceHandle_t** publication_handle)

  *This operation returns data that describes a particular matched **DataWriter** (p. 37) identified by **publication_handle**.*
- **QueryCondition** ∗ **create_querycondition** (SampleStateMask sample_states, ViewStateMask view_states, InstanceStateMask instance_states, const char ∗query_expression, const **StringSeq** &query_parameters)

  *Creates a **QueryCondition** (p. 61).*
- **ReturnCode_t read** (**Sequence** ∗received_data, **SampleInfoSeq** ∗sample_infos, long max_samples, SampleStateMask sample_states, ViewStateMask view_states, InstanceStateMask instance_states)

  *This operation accesses a collection of data values (with associated SampleInfo) within the **DataReader** (p. 26).*
- **ReturnCode_t take** (**Sequence** ∗received_data, **SampleInfoSeq** ∗sample_infos, long max_samples, SampleStateMask sample_states, ViewStateMask view_states, InstanceStateMask instance_states)

  *This operation takes a collection of data values (with associated SampleInfo) from the **DataReader** (p. 26).*
- **ReturnCode_t read_w_condition** (**Sequence** ∗received_data, **SampleInfoSeq** ∗sample_infos, long max_samples, **ReadCondition** ∗a_condition)

  *This operation accesses a collection of data values (with associated SampleInfo), subject to a filter, within the **DataReader** (p. 26).*
- **ReturnCode_t take_w_condition** (**Sequence** ∗received_data, **SampleInfoSeq** ∗sample_infos, long max_samples, **ReadCondition** ∗a_condition)

  *This operation takes a collection of data values (with associated SampleInfo), subject to a filter, from the **DataReader** (p. 26).*
- **ReturnCode_t read_next_sample** (void ∗received_data, SampleInfo ∗sample_info)

  *This operation accesses a data value (with associated SampleInfo) within the **DataReader** (p. 26).*
- **ReturnCode_t take_next_sample** (void ∗received_data, SampleInfo ∗sample_info)

  *This operation takes a data value (with associated SampleInfo) from the **DataReader** (p. 26).*
- **ReturnCode_t read_instance** (**Sequence** ∗received_data, **SampleInfoSeq** ∗sample_infos, long max_samples, **InstanceHandle_t** a_handle, SampleStateMask sample_states, ViewStateMask view_states, InstanceStateMask instance_states)

  *This operation accesses a collection of data values (with associated SampleInfo), belonging to a particular instance, within the **DataReader** (p. 26).*
- **ReturnCode_t take_instance** (**Sequence** ∗received_data, **SampleInfoSeq** ∗sample_infos, long max_samples, **InstanceHandle_t** a_handle, SampleStateMask sample_states, ViewStateMask view_states, InstanceStateMask instance_states)

  *This operation takes a collection of data values (with associated SampleInfo), belonging to a particular instance, from the **DataReader** (p. 26).*

- **ReturnCode_t read_next_instance** (**Sequence** ∗received_data, **SampleInfoSeq** ∗sample_infos, long max_samples, **InstanceHandle_t** previous_handle, SampleStateMask sample_states, ViewStateMask view_states, InstanceStateMask instance_states)

  *This operation accesses a collection of data values (with associated SampleInfo), instance by instance, within the **DataReader** (p. 26).*
- **ReturnCode_t take_next_instance** (**Sequence** ∗received_data, **SampleInfoSeq** ∗sample_infos, long max_samples, **InstanceHandle_t** previous_handle, SampleStateMask sample_states, ViewStateMask view_states, InstanceStateMask instance_states)

  *This operation takes a collection of data values (with associated SampleInfo), instance by instance, from the **DataReader** (p. 26).*
- **ReturnCode_t read_next_instance_w_condition** (**Sequence** ∗received_data, **SampleInfoSeq** ∗sample_infos, long max_samples, **InstanceHandle_t** previous_handle, **ReadCondition** ∗a_condition)

  *This operation accesses a collection of data values (with associated SampleInfo), instance by instance subject to a filter, within the **DataReader** (p. 26).*
- **ReturnCode_t take_next_instance_w_condition** (**Sequence** ∗received_data, **SampleInfoSeq** ∗sample_infos, long max_samples, **InstanceHandle_t** previous_handle, **ReadCondition** ∗a_condition)

  *This operation takes a collection of data values (with associated SampleInfo), instance by instance subject to a filter, from the **DataReader** (p. 26).*
- **ReturnCode_t return_loan** (**Sequence** ∗received_data, **SampleInfoSeq** ∗sample_infos)

  *Returns data and sample_info values to a **DataReader** (p. 26).*
- **ReturnCode_t get_key_value** (void ∗key_holder, **InstanceHandle_t** handle)

  *This routine will populate the data structure indicated by **key_holder** with the key infomation identified by **handle**.*
- **InstanceHandle_t lookup_instance** (void ∗instance_data)

  *Returns the handle that identifies the data instance provided in **instance_data**.*
- **ReturnCode_t get_guid** (**GUID_t** ∗guid)

  Access the GUID which uniquely identifies this reader.

2.3.1 Detailed Description

The **DataReader** (p. 26) entity allows the application to subscribe to and read data.

The DataReadre is an abstract 'class' that is extended to support a particular data type required by the application. A **DataReader** (p. 26) is associated with a single **TopicDescription** (p. 72) (**Topic** (p. 69), MultiTopic, or **ContentFiltered-Topic** (p. 24)).

2.3.2 Member Function Documentation

2.3.2.1 QueryCondition∗ DDS::DataReader::create_querycondition (SampleStateMask *sample_states,* ViewStateMask *view_states,* InstanceStateMask *instance_states,* const char ∗ *query_expression,* const StringSeq & *query_parameters*)

Creates a **QueryCondition** (p. 61).

The returned **QueryCondition** (p. 61) can be used as an argument to **read_w_condition()** (p. 34) or **take_w_condition()** (p. 37).

```
The \b query_expression is an SQL like condition expression, and \b query_parameters
provide optional parameters that are referenced by the \b 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 '<b>LIKE</b>' 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 '[&lt;characters&gt;]' to match a range of characters.

CoreDX %DDS also includes support for the '<b>IN</b>' 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 \b 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

QueryCondition (p. 61)
DataReader_read_w_condition()
DataReader_take_w_condition()

2.3.2.2 ReadCondition* DDS::DataReader::create_readcondition (SampleStateMask *sample_states,* ViewStateMask *view_states,* InstanceStateMask *instance_states*)

Creates a **ReadCondition** (p. 63) that is associated with this **DataReader** (p. 26).

The returned condition can be added to a **WaitSet** (p. 73) or used in a call to the specialized **read() (p. 32)** or **take() (p. 35)** operations. For example see DataReader_read_w_condition().

2.3.2.3 ReturnCode_t DDS::DataReader::delete_contained_entities ( )

This operation deletes all the **ReadCondition** (p. 63) and **QueryCondition** (p. 61) objects previously created by means of the DataReader_create_readcondition() and DataReader_create_querycondition() operations.

After successful execution, the application may delete the **Publisher** (p. 57) by calling Subscriber_delete_datareader().

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

2.3.2.4 ReturnCode_t DDS::DataReader::delete_readcondition (ReadCondition * *a_condition*)

Destroys a **ReadCondition** (p. 63) (or **QueryCondition** (p. 61)).

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** (p. 26) **dr**. Otherwise, the error RETCODE_PRECONDITION_NOT_MET will be returned.

If the **DataReader** (p. 26) is actively processing the **ReadCondition** (p. 63), this routine will return RETCODE_ERROR; in this case, the **delete_readcondition()** (p. 29) call should be re-tried.

2.3.2.5 ReturnCode_t DDS::DataReader::enable ( ) [virtual]

Enables the **DataReader** (p. 26).

A **DataReader** (p. 26) is created either enabled or not based on the **SubscriberQos** (p. 79) setting **entity_factory**. When a **DataReader** (p. 26) is not enabled, only the following sub-set of all **DataReader** (p. 26) operations are legal:

- operations to get and set QoS policies,
- **get_statuscondition()** (p. 26),
- **get_status_changes()** (p. 31),

Any other operation may return the NOT_ENABLED error. DataReader_enable() may be called on an already enabled **DataReader** (p. 26) [it will have no effect].

Reimplemented from **DDS::Entity** (p. 56).

2.3.2.6 ReturnCode_t DDS::DataReader::get_key_value (void * *key_holder,* InstanceHandle_t *handle*)

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

Note

This routine is data type specific. The generated type specific **DataReader** (p. 26) includes an implementation of this routine which should be used to support type-safety.

2.3.2.7 DataReaderListener* DDS::DataReader::get_listener ( )

This operation returns the currently installed **DataReaderListener** (p. 107).

Note

Because the infrastructure makes a copy of the listener provided in DataReader_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.

2.3.2.8 ReturnCode_t DDS::DataReader::get_liveliness_changed_status (LivelinessChangedStatus * *status*)

Provides access to the current LivelinessChangedStatus of the **DataReader** (p. 26).

As a side-effect, this routine will reset the **total_count_change** status field to zero.

2.3.2.9 ReturnCode_t DDS::DataReader::get_matched_publication_data (PublicationBuiltinTopicData * *publication_data,* const InstanceHandle_t *publication_handle*)

This operation returns data that describes a particular matched **DataWriter** (p. 37) 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 matched **DataWriter** (p. 37), this routine will return RETCODE_PRECONDITION_NOT_MET.

2.3.2.10 ReturnCode_t DDS::DataReader::get_matched_publications (InstanceHandleSeq * *publication_handles*)

This operation retrieves the list of DataWriters currently matched with the **DataReader** (p. 26) **dr**.

This list will include the handles that identify DataWriters which have matching **Topic** (p. 69) and compatible QoS with **DataReader** (p. 26).

If a **DataWriter** (p. 37) has been ignored by a call to DomainParticipant_ignore_publication(), then it will not appear in the list.

2.3.2.11 ReturnCode_t DDS::DataReader::get_qos (DataReaderQos & *qos*)

Returns the current **DataReaderQos** (p. 75) settings held in the **DataReader** (p. 26) **dr**.

This routines copies data from the **DataReader** (p. 26) QoS properties into **qos**.

Note

> The qos structure may contain sequences or strings that are populated with dynamic memory. The caller is responsible for freeing the dynamic memory of these items.
> For example, the sequence 'qos->user_data' may have dynamically allocated memory assigned. This can be released by a call to **seq_clear(&qos->user_data)**.

2.3.2.12 ReturnCode_t DDS::DataReader::get_requested_deadline_missed_status (RequestedDeadlineMissedStatus * *status*)

Provides access to the current RequestedDeadlineMissedStatus of the **DataReader** (p. 26).

As a side-effect, this routine will reset the **total_count_change** status field to zero.

2.3.2.13 ReturnCode_t DDS::DataReader::get_requested_incompatible_qos_status (RequestedIncompatibleQosStatus * *status*)

Provides access to the current RequestedIncompatibleQosStatus of the **DataReader** (p. 26).

As a side-effect, this routine will reset the **total_count_change** status field to zero.

2.3.2.14 ReturnCode_t DDS::DataReader::get_sample_lost_status (SampleLostStatus * *status*)

Provides access to the current SampleLostStatus of the **DataReader** (p. 26).

As a side-effect, this routine will reset the **total_count_change** status field to zero.

2.3.2.15 ReturnCode_t DDS::DataReader::get_sample_rejected_status (SampleRejectedStatus * *status*)

Provides access to the current SampleRejectedStatus of the **DataReader** (p. 26).

As a side-effect, this routine will reset the **total_count_change** status field to zero.

2.3.2.16 StatusMask DDS::DataReader::get_status_changes ( ) [virtual]

This returns the list of **triggered** communication statuses in the **DataReader** (p. 26).

If the **DataReader** (p. 26) is not enabled, all statuses will be **untriggered**.

Implements **DDS::Entity** (p. 56).

2.3.2.17 ReturnCode_t DDS::DataReader::get_subscription_matched_status (SubscriptionMatchedStatus * *status*)

Provides access to the current SubscriptionMatchedStatus of the **DataReader** (p. 26).

As a side-effect, this routine will reset the **total_count_change** status field to zero.

2.3.2.18 TopicDescription* DDS::DataReader::get_topicdescription ( )

Returns the **TopicDescription** (p. 72) associated with **DataReader** (p. 26) **dr**.

The returned **TopicDescription** (p. 72) is really a **Topic** (p. 69), **ContentFilteredTopic** (p. 24) or MultiTopic.

2.3.2.19 InstanceHandle_t DDS::DataReader::lookup_instance (void * *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.

Note

This routine is data type specific. The generated type specific **DataReader** (p. 26) includes an implementation of this routine which should be used to support type-safety.

2.3.2.20 ReturnCode_t DDS::DataReader::read (Sequence * *received_data,* SampleInfoSeq * *sample_infos,* long *max_samples,* SampleStateMask *sample_states,* ViewStateMask *view_states,* InstanceStateMask *instance_states*)

This operation accesses a collection of data values (with associated SampleInfo) within the **DataReader** (p. 26).

This routine, and the related **take()** (p. 35), provide the interface for an application to access published data. There are several varieties of **read()** (p. 32) and **take()** (p. 35), to facilitate different access patterns or approaches.

The primary difference between **read()** (p. 32) and **take()** (p. 35) is that **take()** (p. 35) removes all returned data samples from the **DataReader** (p. 26) while **read()** (p. 32) does not. Sequential **read()** (p. 32) calls will return the same data samples each time (if nothing else changes); while sequential **take()** (p. 35) calls will return data samples for only the first call. Subsequent **take()** (p. 35) calls will return an empty collection (if no new data arrives).

The specific behavior of **read()** (p. 32) depends on several things: input paramters, the QoS settings of the **DataReader** (p. 26), 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.: 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 DataReader_return_loan().

The **read() (p. 32)** operation will set the SampleInfo::sample_state to READ_SAMPLE_STATE.

The **read() (p. 32)** operation may set the SampleInfo::view_state to 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() (p. 32)** will return RETCODE_NO_DATA.

Note

This routine is data type specific. The generated type specific **DataReader** (p. 26) includes an implementation of this routine which should be used to support type-safety.

2.3.2.21 ReturnCode_t DDS::DataReader::read_instance (Sequence * *received_data,* SampleInfoSeq * *sample_infos,* long *max_samples,* InstanceHandle_t *a_handle,* SampleStateMask *sample_states,* ViewStateMask *view_states,* InstanceStateMask *instance_states*)

This operation accesses a collection of data values (with associated SampleInfo), belonging to a particular instance, within the **DataReader** (p. 26).

Note

This routine is data type specific. The generated type specific **DataReader** (p. 26) includes an implementation of this routine which should be used to support type-safety.

See also

DataReader_read()

2.3.2.22 ReturnCode_t DDS::DataReader::read_next_instance (Sequence * *received_data,* SampleInfoSeq * *sample_infos,* long *max_samples,* InstanceHandle_t *previous_handle,* SampleStateMask *sample_states,* ViewStateMask *view_states,* InstanceStateMask *instance_states*)

This operation accesses a collection of data values (with associated SampleInfo), instance by instance, within the **DataReader** (p. 26).

Note

This routine is data type specific. The generated type specific **DataReader** (p. 26) includes an implementation of this routine which should be used to support type-safety.

See also

DataReader_read()

2.3.2.23 ReturnCode_t DDS::DataReader::read_next_instance_w_condition (Sequence * *received_data,* SampleInfoSeq * *sample_infos,* long *max_samples,* InstanceHandle_t *previous_handle,* ReadCondition * *a_condition*)

This operation accesses a collection of data values (with associated SampleInfo), instance by instance subject to a filter, within the **DataReader** (p. 26).

Note

This routine is data type specific. The generated type specific **DataReader** (p. 26) includes an implementation of this routine which should be used to support type-safety.

See also

DataReader_read()

2.3.2.24 ReturnCode_t DDS::DataReader::read_next_sample (void * *received_data,* SampleInfo * *sample_info*)

This operation accesses a data value (with associated SampleInfo) within the **DataReader** (p. 26).

Note

This routine is data type specific. The generated type specific **DataReader** (p. 26) includes an implementation of this routine which should be used to support type-safety.

See also

DataReader_read()

2.3.2.25 ReturnCode_t DDS::DataReader::read_w_condition (Sequence * *received_data,* SampleInfoSeq * *sample_infos,* long *max_samples,* ReadCondition * *a_condition*)

This operation accesses a collection of data values (with associated SampleInfo), subject to a filter, within the **DataReader** (p. 26).

Note

This routine is data type specific. The generated type specific **DataReader** (p. 26) includes an implementation of this routine which should be used to support type-safety.

See also

DataReader_read()

2.3.2.26 ReturnCode_t DDS::DataReader::return_loan (Sequence * *received_data,* SampleInfoSeq * *sample_infos*)

Returns data and sample_info values to a **DataReader** (p. 26).

When an application calls **DataReader** (p. 26) **read() (p. 32)** or **take() (p. 35)** operations, these routines may '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** (p. 26). The **return_loan()** (p. 34) routine indicates to the **DataReader** (p. 26) that the application no longer requires access to the data and sample_infos.

A call to **return_loan() (p. 34)** operation must be called only if previous **read() (p. 32)** or **take() (p. 35)** calls 'loaned' data to the application. See DataReader_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** (p. 26) **dr**, then the error RETCODE_PRECONDITION_NOT_MET will be returned.

Note

A **DataReader** (p. 26) cannot be deleted if any 'loans' are outstanding.

2.3.2.27 ReturnCode_t DDS::DataReader::set_listener (DataReaderListener * *a_listener,* StatusMask *mask*)

Installs a **DataReaderListener** (p. 107) on **DataReader** (p. 26) **dr**.

Only one listener may be attached to a **DataReader** (p. 26) at a time. A call to **set_listener() (p. 35)** 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.

2.3.2.28 ReturnCode_t DDS::DataReader::set_qos (const DataReaderQos & *qos*)

Sets the **DataReaderQos** (p. 75) values.

These QoS values affect the behavior of the **DataReader** (p. 26).

2.3.2.29 ReturnCode_t DDS::DataReader::take (Sequence * *received_data,* SampleInfoSeq * *sample_infos,* long *max_samples,* SampleStateMask *sample_states,* ViewStateMask *view_states,* InstanceStateMask *instance_states*)

This operation takes a collection of data values (with associated SampleInfo) from the **DataReader** (p. 26).

This routine, and the related **read() (p. 32)**, provide the interface for an application to access published data. There are several varieties of **read() (p. 32)** and **take() (p. 35)**, to facilitate different access patterns or approaches.

The primary difference between **read() (p. 32)** and **take() (p. 35)** is that **take() (p. 35)** removes all returned data samples from the **DataReader** (p. 26) while **read() (p. 32)** does not. Sequential **read() (p. 32)** calls will return the same data samples each time (if nothing else changes); while sequential **take() (p. 35)** calls will return data samples for only the first call. Subsequent **take() (p. 35)** calls will return an empty collection (if no new data arrives).

The specific behavior of **take() (p. 35)** depends on several things: input paramters, the QoS settings of the **DataReader** (p. 26), 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.: 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 DataReader_return_loan().

The **take() (p. 35)** operation will set the SampleInfo::sample_state to READ_SAMPLE_STATE.

The **take() (p. 35)** operation may set the SampleInfo::view_state to 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() (p. 35)** will return RETCODE_NO_DATA.

Note

This routine is data type specific. The generated type specific **DataReader** (p. 26) includes an implementation of this routine which should be used to support type-safety.

2.3.2.30 ReturnCode_t DDS::DataReader::take_instance (Sequence * *received_data,* SampleInfoSeq * *sample_infos,* long *max_samples,* InstanceHandle_t *a_handle,* SampleStateMask *sample_states,* ViewStateMask *view_states,* InstanceStateMask *instance_states*)

This operation takes a collection of data values (with associated SampleInfo), belonging to a particular instance, from the **DataReader** (p. 26).

Note

This routine is data type specific. The generated type specific **DataReader** (p. 26) includes an implementation of this routine which should be used to support type-safety.

See also

DataReader_take()

2.3.2.31 ReturnCode_t DDS::DataReader::take_next_instance (Sequence * *received_data,* SampleInfoSeq * *sample_infos,* long *max_samples,* InstanceHandle_t *previous_handle,* SampleStateMask *sample_states,* ViewStateMask *view_states,* InstanceStateMask *instance_states*)

This operation takes a collection of data values (with associated SampleInfo), instance by instance, from the **DataReader** (p. 26).

Note

This routine is data type specific. The generated type specific **DataReader** (p. 26) includes an implementation of this routine which should be used to support type-safety.

See also

DataReader_take()

2.3.2.32 ReturnCode_t DDS::DataReader::take_next_instance_w_condition (Sequence * *received_data,* SampleInfoSeq * *sample_infos,* long *max_samples,* InstanceHandle_t *previous_handle,* ReadCondition * *a_condition*)

This operation takes a collection of data values (with associated SampleInfo), instance by instance subject to a filter, from the **DataReader** (p. 26).

Note

This routine is data type specific. The generated type specific **DataReader** (p. 26) includes an implementation of this routine which should be used to support type-safety.

See also

DataReader_take()

2.3.2.33 ReturnCode_t DDS::DataReader::take_next_sample (void * *received_data,* SampleInfo * *sample_info*)

This operation takes a data value (with associated SampleInfo) from the **DataReader** (p. 26).

Note

This routine is data type specific. The generated type specific **DataReader** (p. 26) includes an implementation of this routine which should be used to support type-safety.

See also

DataReader_take()

2.3.2.34 ReturnCode_t DDS::DataReader::take_w_condition (Sequence * *received_data,* SampleInfoSeq * *sample_infos,* long *max_samples,* ReadCondition * *a_condition*)

This operation takes a collection of data values (with associated SampleInfo), subject to a filter, from the **DataReader** (p. 26).

Note

This routine is data type specific. The generated type specific **DataReader** (p. 26) includes an implementation of this routine which should be used to support type-safety.

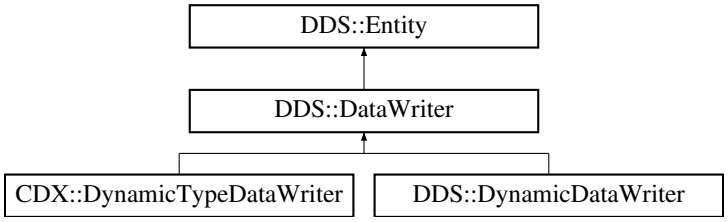
See also

DataReader_take()

2.4 DDS::DataWriter Class Reference

The **DataWriter** (p. 37) entity provides an interface for the application to publish (write) data.

Inheritance diagram for DDS::DataWriter:



Public Member Functions

- **ReturnCode_t enable** ()

  *Enables the **DataWriter** (p. 37).*
- **InstanceHandle_t get_instance_handle** ()

  *This operation returns the InstanceHandle_t that identifies the **DataWriter** (p. 37).*
- **StatusMask get_status_changes** ()

*This returns the list of **triggered** communication statuses in the **DataWriter** (p. 37).*

- **ReturnCode_t set_qos** (const **DataWriterQos** &qos)

  *Sets the **DataWriterQos** (p. 76) values.*
- **ReturnCode_t get_qos** (**DataWriterQos** &qos)

  *Returns the current **DataWriterQos** (p. 76) settings held in the **DataWriter** (p. 37) **dw**.*
- **ReturnCode_t set_listener** (**DataWriterListener** ∗a_listener, **StatusMask** mask)

  *Installs a **DataWriterListener** (p. 109) on **DataWriter** (p. 37) **dw**.*
- **DataWriterListener** ∗ **get_listener** ()

  *This operation returns the currently installed **DataWriterListener** (p. 109).*
- class **Topic** ∗ **get_topic** ()

  *Returns the **Topic** (p. 69) associated with **DataWriter** (p. 37) **dw**.*
- class **Publisher** ∗ **get_publisher** ()

  *Returns the **Publisher** (p. 57) that contains **DataWriter** (p. 37) **dw**.*
- **ReturnCode_t wait_for_acknowledgments** (const **Duration_t** &max_wait)

  Block until this writer has received acknowledgements for all written data.
- **ReturnCode_t assert_liveliness** ()

  *This operation manually asserts the liveliness of the **DataWriter** (p. 37) **dw**.*
- **InstanceHandle_t register_instance** (const void ∗instance_data)

  *Declares the existance of an instance identified by the 'key fields' in **instance_data**.*
- **InstanceHandle_t register_instance_w_timestamp** (const void ∗instance_data, const **Time_t** &source_timestamp)

  *Declares the existance of an instance identified by the 'key fields' in **instance_data**.*
- **ReturnCode_t unregister_instance** (const void ∗instance_data, const **InstanceHandle_t** handle)

  Indicates that the writer will no longer be providing updates the specified instance.
- **ReturnCode_t unregister_instance_w_timestamp** (const void ∗instance_data, const **InstanceHandle_t** handle, const **Time_t** &source_timestamp)

  Indicates that the writer will no longer be providing updates the specified instance.
- **ReturnCode_t write** (const void ∗instance_data, const **InstanceHandle_t** handle)

  *Publishes the provided **instance_data**.*
- **ReturnCode_t write_w_timestamp** (const void ∗instance_data, const **InstanceHandle_t** handle, const **Time_t** &source_timestamp)

  *Publishes the provided **instance_data**.*
- **ReturnCode_t dispose** (const void ∗instance_data, const **InstanceHandle_t** instance_handle)

  Indicates that the instance no longer exists.
- **ReturnCode_t dispose_w_timestamp** (const void ∗instance_data, const **InstanceHandle_t** instance_handle, const **Time_t** &source_timestamp)

  Indicates that the instance no longer exists.
- **ReturnCode_t get_key_value** (void ∗key_holder, const **InstanceHandle_t** handle)

  *This routine will populate the data structure indicated by **key_holder** with the key infomation identified by **handle**.*
- **InstanceHandle_t lookup_instance** (const void ∗instance_data)

  *Returns the handle that identifies the data instance provided in **instance_data**.*
- **ReturnCode_t write_request** (const void ∗instance_data, const **InstanceHandle_t** h, **DDS::SampleIdentity_t** ∗req_id)

  This method is used by the RPC implementation to send a request sample and record the sample identity.
- **ReturnCode_t write_reply** (const void ∗instance_data, const **InstanceHandle_t** h, **DDS::SampleIdentity_t** ∗rep_id)

  This method is used by the RPC implementation to send a reply sample populated with the related request sample identity.
- **ReturnCode_t get_liveliness_lost_status** (LivelinessLostStatus ∗status)

  *Provides access to the current LivelinessLostStatus of the **DataWriter** (p. 37).*

- **ReturnCode_t get_offered_deadline_missed_status** (OfferedDeadlineMissedStatus *status)

  *Provides access to the current OfferedDeadlineMissedStatus of the **DataWriter** (p. 37).*
- **ReturnCode_t get_offered_incompatible_qos_status** (OfferedIncompatibleQosStatus *status)

  *Provides access to the current OfferedIncompatibleQosStatus of the **DataWriter** (p. 37).*
- **ReturnCode_t get_publication_matched_status** (PublicationMatchedStatus *status)

  *Provides access to the current PublicationMatchedStatus of the **DataWriter** (p. 37).*
- **ReturnCode_t get_matched_subscriptions** (**InstanceHandleSeq** *subscription_handles)

  *This operation retrieves the list of DataReaders currently matched with the **DataWriter** (p. 37) **dw**.*
- **ReturnCode_t get_matched_subscription_data** (**SubscriptionBuiltinTopicData** *subscription_data, **InstanceHandle_t** &subscription_handle)

  *This operation returns data that describes a particular matched **DataReader** (p. 26) identified by **subscription_handle**.*
- **ReturnCode_t get_guid** (**GUID_t** *guid)

  Access the GUID which uniquely identifies this writer.

2.4.1 Detailed Description

The **DataWriter** (p. 37) entity provides an interface for the application to publish (write) data.

The **DataWriter** (p. 37) is an abstract 'class' that is extended to support a particular data type required by the application. A **DataReader** (p. 26) is associated with, and writes on, a single **Topic** (p. 69).

2.4.2 Member Function Documentation

2.4.2.1 ReturnCode_t DDS::DataWriter::assert_liveliness ( )

This operation manually asserts the liveliness of the **DataWriter** (p. 37) **dw**.

This operation is useful if the LIVELINESS QoS setting is MANUAL_BY_PARTICIPANT or MANUAL_BY_TOPIC; otherwise, it has no effect.

Note

The **write** operation automatically asserts livelines on the **DataWriter** (p. 37) and its **DomainParticipant** (p. 44). Therefore, **assert_liveliness** is required only if hte application is not writing data frequently enough to satisfy the LIVELINESS setting.

2.4.2.2 ReturnCode_t DDS::DataWriter::dispose (const void * *instance_data,* const InstanceHandle_t *instance_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** (p. 37). The current time is used as the timstamp.

2.4.2.3 ReturnCode_t DDS::DataWriter::dispose_w_timestamp (const void * *instance_data,* const InstanceHandle_t *instance_handle,* const Time_t & *source_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** (p. 37). The **source_timestamp** is used as the source timstamp for the published message.

2.4.2.4 ReturnCode_t DDS::DataWriter::enable ( ) [virtual]

Enables the **DataWriter** (p. 37).

A **DataWriter** (p. 37) is created either enabled or not based on the **PublisherQos** (p. 78) setting **entity_factory**. When a **DataWriter** (p. 37) is not enabled, only the following sub-set of all **DataWriter** (p. 37) operations are legal:

- operations to get and set QoS policies,
- **get_statuscondition()** (p. 56),
- **get_status_changes()** (p. 41),

Any other operation may return the NOT_ENABLED error. DataWriter_enable() may be called on an already enabled **DataWriter** (p. 37) [it will have no effect].

Reimplemented from **DDS::Entity** (p. 56).

2.4.2.5 ReturnCode_t DDS::DataWriter::get_key_value (void * *key_holder,* const InstanceHandle_t *handle*)

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

Note

This routine is data type specific. The generated type specific **DataWriter** (p. 37) includes an implementation of this routine which should be used to support type-safety.

2.4.2.6 DataWriterListener* DDS::DataWriter::get_listener ( )

This operation returns the currently installed **DataWriterListener** (p. 109).

Note

Because the infrastructure makes a copy of the listener provided in DataWriter_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.

2.4.2.7 ReturnCode_t DDS::DataWriter::get_liveliness_lost_status (LivelinessLostStatus * *status*)

Provides access to the current LivelinessLostStatus of the **DataWriter** (p. 37).

As a side-effect, this routine will reset the **total_count_change** status field to zero.

2.4.2.8 ReturnCode_t DDS::DataWriter::get_matched_subscription_data (SubscriptionBuiltinTopicData * *subscription_data,* InstanceHandle_t & *subscription_handle*)

This operation returns data that describes a particular matched **DataReader** (p. 26) 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** (p. 26), this routine will return RETCODE_PRECONDITION_NOT_MET.

2.4.2.9 ReturnCode_t DDS::DataWriter::get_matched_subscriptions (InstanceHandleSeq * *subscription_handles*)

This operation retrieves the list of DataReaders currently matched with the **DataWriter** (p. 37) **dw**.

This list will include the handles that identify DataReaders which have matching **Topic** (p. 69) and compatible QoS with **DataWriter** (p. 37).

If a **DataReader** (p. 26) has been ignored by a call to DomainParticipant_ignore_subscription(), then it will not appear in the list.

2.4.2.10 ReturnCode_t DDS::DataWriter::get_offered_deadline_missed_status (OfferedDeadlineMissedStatus * *status*)

Provides access to the current OfferedDeadlineMissedStatus of the **DataWriter** (p. 37).

As a side-effect, this routine will reset the **total_count_change** status field to zero.

2.4.2.11 ReturnCode_t DDS::DataWriter::get_offered_incompatible_qos_status (OfferedIncompatibleQosStatus * *status*)

Provides access to the current OfferedIncompatibleQosStatus of the **DataWriter** (p. 37).

As a side-effect, this routine will reset the **total_count_change** status field to zero.

2.4.2.12 ReturnCode_t DDS::DataWriter::get_publication_matched_status (PublicationMatchedStatus * *status*)

Provides access to the current PublicationMatchedStatus of the **DataWriter** (p. 37).

As a side-effect, this routine will reset the **total_count_change** and **current_count_change** status fields to zero.

2.4.2.13 ReturnCode_t DDS::DataWriter::get_qos (DataWriterQos & *qos*)

Returns the current **DataWriterQos** (p. 76) settings held in the **DataWriter** (p. 37) **dw**.

This routines copies data from the **DataWriter** (p. 37) QoS properties into **qos**.

Note

> The qos structure may contain sequences or strings that are populated with dynamic memory. The caller is responsible for freeing the dynamic memory of these items.
> For example, the sequence 'qos->user_data' may have dynamically allocated memory assigned. This can be released by a call to **seq_clear(&qos->user_data)**.

2.4.2.14 StatusMask DDS::DataWriter::get_status_changes ( ) [virtual]

This returns the list of **triggered** communication statuses in the **DataWriter** (p. 37).

If the **DataWriter** (p. 37) is not enabled, all statuses will be **untriggered**.

Implements **DDS::Entity** (p. 56).

2.4.2.15 InstanceHandle_t DDS::DataWriter::lookup_instance (const void * *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.

2.4.2.16 InstanceHandle_t DDS::DataWriter::register_instance (const void * *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 timestamp.

2.4.2.17 InstanceHandle_t DDS::DataWriter::register_instance_w_timestamp (const void * *instance_data,* const Time_t & *source_timestamp*)

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 timestamp.

2.4.2.18 ReturnCode_t DDS::DataWriter::set_listener (DataWriterListener * *a_listener,* StatusMask *mask*)

Installs a **DataWriterListener** (p. 109) on **DataWriter** (p. 37) **dw**.

Only one listener may be attached to a **DataWriter** (p. 37) at a time. A call to **set_listener() (p. 42)** 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.

2.4.2.19 ReturnCode_t DDS::DataWriter::set_qos (const DataWriterQos & *qos*)

Sets the **DataWriterQos** (p. 76) values.

These QoS values affect the behavior of the **DataWriter** (p. 37).

2.4.2.20 ReturnCode_t DDS::DataWriter::unregister_instance (const void * *instance_data,* const 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** (p. 37). The current time is used as the timstamp.

2.4.2.21 ReturnCode_t DDS::DataWriter::unregister_instance_w_timestamp (const void * *instance_data,* const InstanceHandle_t *handle,* const Time_t & *source_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** (p. 37). The provided **source_timestamp** is used as the timstamp.

2.4.2.22 ReturnCode_t DDS::DataWriter::wait_for_acknowledgments (const Duration_t & *max_wait*)

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

RETCODE_TIME_OUT	returned if 'max_wait' passes before all acks are received

Return values

RETCODE_OK	returned if all acks have been received before 'max_wait'

2.4.2.23 ReturnCode_t DDS::DataWriter::write (const void * *instance_data,* const 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.

2.4.2.24 ReturnCode_t DDS::DataWriter::write_w_timestamp (const void * *instance_data,* const InstanceHandle_t *handle,* const Time_t & *source_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.

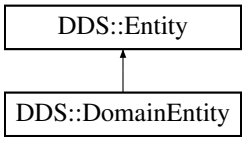
The routine may return RETCODE_OUT_OF_RESOURCES if it is determined that no amount of blocking will allow the data to be processed.

On success, RETCODE_OK is returned.

2.5 DDS::DomainEntity Class Reference

Base class for all DDS Domain Entities.

Inheritance diagram for DDS::DomainEntity:



Additional Inherited Members

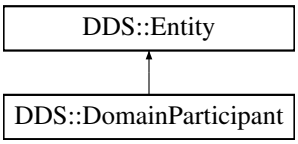
2.5.1 Detailed Description

Base class for all DDS Domain Entities.

2.6 DDS::DomainParticipant Class Reference

The **DomainParticipant** (p. 44) is used to configure, create and destroy **Publisher** (p. 57), **Subscriber** (p. 65) and **Topic** (p. 69) objects.

Inheritance diagram for DDS::DomainParticipant:



Public Member Functions

- **ReturnCode_t get_qos** (**DomainParticipantQos** &qos)

  *Provides access to the DomainParticipantQoS settings of the **DomainParticipant** (p. 44).*
- **ReturnCode_t set_qos** (const **DomainParticipantQos** &qos)

  Sets the DomainParticipantQoS values.
- **StatusMask get_status_changes** ()

  *This returns the list of **triggered** communication statuses in the **DomainParticipant** (p. 44).*
- **ReturnCode_t set_listener** (**DomainParticipantListener** ∗a_listener, **StatusMask** mask)

  *This operation installs a **DomainParticipantListener** (p. 110) on the **DomainParticipant** (p. 44).*
- **DomainParticipantListener** ∗ **get_listener** ()

  *This operation returns the currently installed **DomainParticipantListener** (p. 110).*
- class **Publisher** ∗ **create_publisher** (const **PublisherQos** &qos, **PublisherListener** ∗a_listener, **StatusMask** mask)

  *This operation creates a **Publisher** (p. 57) with the specified PublisherQoS settings and **PublisherListener** (p. 110).*
- **ReturnCode_t delete_publisher** (class **Publisher** ∗p)

  *This operation deletes an existing **Publisher** (p. 57).*
- class **Subscriber** ∗ **create_subscriber** (const **SubscriberQos** &qos, **SubscriberListener** ∗a_listener, **StatusMask** mask)

  *This operation creates a **Subscriber** (p. 65) with the specified **SubscriberQos** (p. 79) and **SubscriberListener** (p. 111).*
- **ReturnCode_t delete_subscriber** (class **Subscriber** ∗s)

  *This operation deletes an existing **Subscriber** (p. 65).*
- class **Topic** ∗ **create_topic** (const char ∗topic_name, const char ∗type_name, const **TopicQos** &qos, **TopicListener** ∗a_listener, **StatusMask** mask)

  *This operation creates a **Topic** (p. 69) with the specified **TopicQos** (p. 80) settings and **TopicListener** (p. 112).*
- **ReturnCode_t delete_topic** (class **Topic** ∗a_topic)

  *This operation deletes a **Topic** (p. 69).*
- class **Topic** ∗ **find_topic** (const char ∗topic_name, const **Duration_t** &timeout)

  *This operation returns a **Topic** (p. 69) identified by the provided **topic_name**.*
- class **TopicDescription** ∗ **lookup_topicdescription** (const char ∗name)

  *This operation returns an existing, locally-created **TopicDescription** (p. 72), named **name**.*
- class **Subscriber** ∗ **get_builtin_subscriber** ()

  *Returns the built-in **Subscriber** (p. 65).*
- **ReturnCode_t ignore_participant** (**InstanceHandle_t** &handle)

  *Instructs the **DomainParticipant** (p. 44) **dp** to ignore the external **DomainParticipant** (p. 44) identified by **handle**.*

- **ReturnCode_t ignore_topic** (**InstanceHandle_t** &handle)

  *This operation instructs the **DomainParticipant** (p. 44) **dp** to ignore a **Topic** (p. 69) identified by **handle**.*
- **ReturnCode_t ignore_publication** (**InstanceHandle_t** &handle)

  *This operation instructs the **DomainParticipant** (p. 44) **dp** to ignore a Publication identified by **handle**.*
- **ReturnCode_t ignore_subscription** (**InstanceHandle_t** &handle)

  *This operation instructs the **DomainParticipant** (p. 44) **dp** to ignore a Subscription identified by **handle**.*
- **DomainId_t get_domain_id** ()

  *This operation retrieves the domain_id to which the **DomainParticipant** (p. 44) belongs.*
- **ReturnCode_t delete_contained_entities** ()

  *This operation deletes all the objects created by means of the **create** operations on **DomainParticipant** (p. 44) **dp**.*
- **ReturnCode_t assert_liveliness** ()

  *This operation manually asserts the liveliness of the **DomainParticipant** (p. 44) **dp**.*
- **ReturnCode_t set_default_publisher_qos** (const **PublisherQos** &qos)

  *Sets the default **PublisherQos** (p. 78) held in the **DomainParticipant** (p. 44).*
- **ReturnCode_t get_default_publisher_qos** (**PublisherQos** &qos)

  *Provides access to the default **PublisherQos** (p. 78) settings held in the factory.*
- **ReturnCode_t set_default_subscriber_qos** (const **SubscriberQos** &qos)

  *Sets the default **SubscriberQos** (p. 79) held in the **DomainParticipant** (p. 44).*
- **ReturnCode_t get_default_subscriber_qos** (**SubscriberQos** &qos)

  *Provides access to the default **SubscriberQos** (p. 79) settings held in the factory.*
- **ReturnCode_t set_default_topic_qos** (const **TopicQos** &qos)

  *Sets the default **TopicQos** (p. 80) held in the **DomainParticipant** (p. 44).*
- **ReturnCode_t get_default_topic_qos** (**TopicQos** &qos)

  *Provides access to the default **TopicQos** (p. 80) settings held in the factory.*
- **ReturnCode_t get_discovered_participants** (**InstanceHandleSeq** ∗participant_handles)

  *This operation returns the list of handles identifying the **DomainParticipant** (p. 44) objects that have been discovered.*
- **ReturnCode_t get_discovered_participant_data** (**ParticipantBuiltinTopicData** ∗participant_data, **InstanceHandle_t** &participant_handle)

  *This operation returns data that describes a particular discovered participant identified by **participant_handle**.*
- **ReturnCode_t get_discovered_topics** (**InstanceHandleSeq** ∗topic_handles)

  *This operation returns the list of handles identifying the **Topic** (p. 69) objects that have been discovered.*
- class **ContentFilteredTopic** ∗ **create_contentfilteredtopic** (const char ∗name, class **Topic** ∗related_topic, const char ∗filter_expression, const **StringSeq** &filter_parameters)

  *This operation creates a **ContentFilteredTopic** (p. 24).*
- **ReturnCode_t delete_contentfilteredtopic** (class **ContentFilteredTopic** ∗a_cft)

  *This operation deletes a previously allocated **ContentFilteredTopic** (p. 24).*
- class MultiTopic ∗ **create_multitopic** (const char ∗name, const char ∗type_name, const char ∗subscription_expression, const **StringSeq** &expression_params)
- **ReturnCode_t delete_multitopic** (class MultiTopic ∗a_multitopic)
- **ReturnCode_t get_discovered_topic_data** (TopicBuiltinTopicData ∗topic_data, **InstanceHandle_t** &topic_handle)

  *This operation returns data that describes a particular discovered topic identified by **topic_handle**.*
- bool **contains_entity** (**InstanceHandle_t** a_handle)

  *This operation checks whether or not the given handle **a_handle** represents an object that was created by the **DomainParticipant** (p. 44) **d**.*
- **ReturnCode_t get_current_time** (**Time_t** ∗current_time)

  This operation returns the current value of the time used by the service.
- **ReturnCode_t do_work** (const **Duration_t** &time_slice)

  *Transfer control to the **DomainParticipant** (p. 44) so it can do background processing.*

- **ReturnCode_t builtin_wait_for_acknowledgments** (const **Duration_t** &max_wait)

  Ensure that local discovery information has been received by all known peers.
- **ReturnCode_t get_guid** (**GUID_t** ∗guid)

  Access the GUID which uniquely identifies this participant.
- **ReturnCode_t enable** ()

  *Enables the **DomainParticipant** (p. 44).*
- **InstanceHandle_t get_instance_handle** ()

  *This operation returns the InstanceHandle_t that identifies the **DomainParticipant** (p. 44).*
- **ReturnCode_t register_type** (TypeSupport ∗ts, const char ∗type_name)

  Registers a TypeSupport with the Participant.
- int **check_version** (const char ∗major, const char ∗minor, const char ∗patch)

  Tests the provided version information against the version of the running CoreDX DDS library.
- void **print_stats** ()

  Prints simple summary debug stats to stdout for each Reader and Writer.

2.6.1 Detailed Description

The **DomainParticipant** (p. 44) is used to configure, create and destroy **Publisher** (p. 57), **Subscriber** (p. 65) and **Topic** (p. 69) objects.

The **DomainParticipant** (p. 44) is a container for the objects that it creates. The objects operate within the **domain** identified by the **domain_id** provided when the **DomainParticipant** (p. 44) was created. Objects within different **domains** do not communicate or interfere with each other.

2.6.2 Member Function Documentation

2.6.2.1 ReturnCode_t DDS::DomainParticipant::assert_liveliness ( )

This operation manually asserts the liveliness of the **DomainParticipant** (p. 44) **dp**.

This operation indicates that the **DomainParticipant** (p. 44) is still alive. It is effective only if the **DomainParticipant** (p. 44) contains on ore more **DataWriter** (p. 37) objects with LIVELINES QoS set to MANUAL_BY_PARTICIPANT. In this case, the operation will assert the liveliness of those particular **DataWriter** (p. 37) objects.

Note

Writing data via the **DataWriter** (p. 37) **write** operation automatically asserts the liveliness of the **DataWriter** (p. 37) and its containing **DomainParticipant** (p. 44). Therefore, the use of **DomainParticipant::assert_liveliness()** (p. 46) is needed only if the application is not writing data frequently enough to keep the **DataWriter** (p. 37) considered 'alive'.

2.6.2.2 int DDS::DomainParticipant::check_version (const char ∗ *major,* const char ∗ *minor,* const char ∗ *patch*)

Tests the provided version information against the version of the running CoreDX DDS library.

If the provided major and minor numbers match those of the running library, then the routine returns 0. If the provided numbers are larger than those of the running library, then the routine returns a positive number (indicating that the provided version is greater than the library version). Finaly, if the provied version is less than the library version, a negative number is returned. Currently, the 'patch' value is ignored.

2.6.2.3 bool DDS::DomainParticipant::contains_entity (InstanceHandle_t *a_handle*)

This operation checks whether or not the given handle **a_handle** represents an object that was created by the **DomainParticipant** (p. 44) **d**.

This applies recursively to all objects created by the **DomainParticipant** (p. 44).

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

2.6.2.4 class ContentFilteredTopic* DDS::DomainParticipant::create_contentfilteredtopic (const char * *name,* class Topic * *related_topic,* const char * *filter_expression,* const StringSeq & *filter_parameters*)

This operation creates a **ContentFilteredTopic** (p. 24).

The **ContentFilteredTopic** (p. 24) 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** (p. 26) associated with the **ContentFilteredTopic** (p. 24).

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.

2.6.2.5 class MultiTopic* DDS::DomainParticipant::create_multitopic (const char * *name,* const char * *type_name,* const char * *subscription_expression,* const StringSeq & *expression_params*)

Not Yet Supported This operation is not implemented yet.

2.6.2.6 class Publisher* DDS::DomainParticipant::create_publisher (const PublisherQos & *qos,* PublisherListener * *a_listener,* StatusMask *mask*)

This operation creates a **Publisher** (p. 57) with the specified PublisherQoS settings and **PublisherListener** (p. 110).

The value **PUBLISHER_QOS_DEFAULT** may be provided for the **qos** argument. This will indicate that the the default publisher QoS settings held in the **DomainParticipant** (p. 44) should be used.

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

2.6.2.7 class Subscriber* DDS::DomainParticipant::create_subscriber (const SubscriberQos & *qos,* SubscriberListener * *a_listener,* StatusMask *mask*)

This operation creates a **Subscriber** (p. 65) with the specified **SubscriberQos** (p. 79) and **SubscriberListener** (p. 111).

The value **SUBSCRIBER_QOS_DEFAULT** may be provided for the **qos** argument. This will indicate that the the default subscriber QoS settings held in the **DomainParticipant** (p. 44) should be used.

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

2.6.2.8 class Topic* DDS::DomainParticipant::create_topic (const char * *topic_name,* const char * *type_name,* const TopicQos & *qos,* TopicListener * *a_listener,* StatusMask *mask*)

This operation creates a **Topic** (p. 69) with the specified **TopicQos** (p. 80) settings and **TopicListener** (p. 112).

The value **TOPIC_QOS_DEFAULT** may be provided for the **qos** argument. This will indicate that the the default topic QoS settings held in the **DomainParticipant** (p. 44) 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** (p. 44) (by a call to **register_type() (p. 53)** on the appropriate **TypeSupport** object) prior to calling **create_topic() (p. 48)**.

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()** (p. 49).

2.6.2.9 ReturnCode_t DDS::DomainParticipant::delete_contained_entities ( )

This operation deletes all the objects created by means of the **create** operations on **DomainParticipant** (p. 44) **dp**.

This routine will recursively call the corresponding **delete_contained_entities()** (p. 48) 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** (p. 44). After successful execution, the application may delete the **DomainParticipant** (p. 44) by calling DomainParticipantFactory_delete_participant().

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

2.6.2.10 ReturnCode_t DDS::DomainParticipant::delete_contentfilteredtopic (class ContentFilteredTopic * *a_cft*)

This operation deletes a previously allocated **ContentFilteredTopic** (p. 24).

This operation will fail if there are any **DataReader** (p. 26), **DataWriter** (p. 37), **ContentFilteredTopic** (p. 24), or MultiTopic objects that reference the specified **Topic** (p. 69). In this case, the operation will return RETCODE_PRECONDITION_NOT_MET.

The **Topic** (p. 69) must be owned by DomainPartipant **dp**; otherwise RETCODE_PRECONDITION_NOT_MET will be returned.

2.6.2.11 ReturnCode_t DDS::DomainParticipant::delete_multitopic (class MultiTopic * *a_multitopic*)

Not Yet Supported This operation is not implemented yet.

2.6.2.12 ReturnCode_t DDS::DomainParticipant::delete_publisher (class Publisher * *p*)

This operation deletes an existing **Publisher** (p. 57).

A **Publisher** (p. 57) cannot be deleted if it contains any **DataWriter** (p. 37) objects. In this case, **delete_publisher** will return RETCODE_PRECONDITION_NOT_MET.

If the **DomainParticipant** (p. 44) **dp** does not contain the provided **Publisher** (p. 57) **p**, the operation will return **RETCODE_PRECONDITION_NOT_MET**.

2.6.2.13 ReturnCode_t DDS::DomainParticipant::delete_subscriber (class Subscriber * *s*)

This operation deletes an existing **Subscriber** (p. 65).

A **Subscriber** (p. 65) cannot be deleted if it contains any **DataReader** (p. 26) objects. In this case, **delete_subscriber** will return RETCODE_PRECONDITION_NOT_MET.

If the **DomainParticipant** (p. 44) **dp** does not contain the provided **Subscriber** (p. 65) **s**, the operation will return **RETCODE_PRECONDITION_NOT_MET**.

2.6.2.14 ReturnCode_t DDS::DomainParticipant::delete_topic (class Topic * *a_topic*)

This operation deletes a **Topic** (p. 69).

This operation will fail if there are any **DataReader** (p. 26), **DataWriter** (p. 37), **ContentFilteredTopic** (p. 24), or MultiTopic objects that reference the specified **Topic** (p. 69). In this case, the operation will return RETCODE_PRECONDITION_NOT_MET.

The **Topic** (p. 69) must be owned by DomainPartipant **dp**; otherwise RETCODE_PRECONDITION_NOT_MET will be returned.

2.6.2.15 ReturnCode_t DDS::DomainParticipant::do_work (const Duration_t & *time_slice*)

Transfer control to the **DomainParticipant** (p. 44) so it can do background processing.

This function is to be called only if the **DomainParticipant** (p. 44) 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** (p. 44) has threads (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.

2.6.2.16 ReturnCode_t DDS::DomainParticipant::enable ( ) [virtual]

Enables the **DomainParticipant** (p. 44).

A **DomainParticipant** (p. 44) 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** (p. 44) operations are legal:

- operations to get and set QoS policies,

- factory operations (create, delete),
- **get_statuscondition()** (p. 56),
- **get_status_changes()** (p. 51),
- lookup operations

Any other **DomainParticipant** (p. 44) operation will return the NOT_ENABLED error. **DomainParticipant::enable()** (p. 49) may be called on an already enabled **DomainParticipant** (p. 44) [it will have no effect].

Reimplemented from **DDS::Entity** (p. 56).

2.6.2.17 class Topic* DDS::DomainParticipant::find_topic (const char * *topic_name,* const Duration_t & *timeout*)

This operation returns a **Topic** (p. 69) 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()** (p. 48), the topic returned from this operation (if not NULL) must be deleted by calling **DomainParticipant::delete_topic()** (p. 49).

2.6.2.18 class Subscriber* DDS::DomainParticipant::get_builtin_subscriber ( )

Returns the built-in **Subscriber** (p. 65).

Each **DomainParticipant** (p. 44) contains several built-in **Topic** (p. 69) objects as well as corresponding **DataReader** (p. 26) objects to access them. These built-in **DataReader** (p. 26) objects belong to the single built-in **Subscriber** (p. 65) that is accessed through this operation.

The built-in Topics are used to communicate information about other **DomainParticipant** (p. 44), **Topic** (p. 69), **DataReader** (p. 26), and **DataWriter** (p. 37) objects.

An application should not explicitly delete the **Subscriber** (p. 65) returned by this operation - it is managed internally.

2.6.2.19 ReturnCode_t DDS::DomainParticipant::get_default_publisher_qos (PublisherQos & *qos*)

Provides access to the default **PublisherQos** (p. 78) settings held in the factory.

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

2.6.2.20 ReturnCode_t DDS::DomainParticipant::get_default_subscriber_qos (SubscriberQos & *qos*)

Provides access to the default **SubscriberQos** (p. 79) settings held in the factory.

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

2.6.2.21 ReturnCode_t DDS::DomainParticipant::get_default_topic_qos (TopicQos & *qos*)

Provides access to the default **TopicQos** (p. 80) settings held in the factory.

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

2.6.2.22 ReturnCode_t DDS::DomainParticipant::get_discovered_participant_data (ParticipantBuiltinTopicData * *participant_data,* InstanceHandle_t & *participant_handle*)

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()** (p. 51).

If **participant_handle** does not identify a known **DomainParticipant** (p. 44), this routine will return RETCODE_PRECONDITION_NOT_MET.

2.6.2.23 ReturnCode_t DDS::DomainParticipant::get_discovered_participants (InstanceHandleSeq * *participant_handles*)

This operation returns the list of handles identifying the **DomainParticipant** (p. 44) 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()** (p. 52).

This routine allocates memory to populate the **participant_handles** sequence. The application is responsible for freeing this memory.

2.6.2.24 ReturnCode_t DDS::DomainParticipant::get_discovered_topic_data (TopicBuiltinTopicData * *topic_data,* InstanceHandle_t & *topic_handle*)

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()** (p. 51).

If **topic_handle** does not identify a known **Topic** (p. 69), this routine will return RETCODE_PRECONDITION_NOT_MET.

2.6.2.25 ReturnCode_t DDS::DomainParticipant::get_discovered_topics (InstanceHandleSeq * *topic_handles*)

This operation returns the list of handles identifying the **Topic** (p. 69) 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()** (p. 52).

This routine allocates memory to populate the **topic_handles** sequence. The application is responsible for freeing this memory.

2.6.2.26 DomainParticipantListener* DDS::DomainParticipant::get_listener ( )

This operation returns the currently installed **DomainParticipantListener** (p. 110).

Note

Because the infrastructure makes a copy of the listener provided in **DomainParticipant::set_listener()** (p. 53), 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.

2.6.2.27 StatusMask DDS::DomainParticipant::get_status_changes ( ) [virtual]

This returns the list of **triggered** communication statuses in the **DomainParticipant** (p. 44).

If the **DomainParticipant** (p. 44) is not enabled, all statuses will be **untriggered**. The list returned by **get_status_changes** may be empty. The list of statuses returned by **get_status_changes** operation contains statuses that are triggered on the **DomainParticipant** (p. 44) itself and does not include statuses from contained objects.

Implements **DDS::Entity** (p. 56).

2.6.2.28 ReturnCode_t DDS::DomainParticipant::ignore_participant (InstanceHandle_t & *handle*)

Instructs the **DomainParticipant** (p. 44) **dp** to ignore the external **DomainParticipant** (p. 44) 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 (p. 137)** via the appropriate built-in **DataReader** (p. 26).

There is no mechanism to reverse this operation.

2.6.2.29 ReturnCode_t DDS::DomainParticipant::ignore_publication (InstanceHandle_t & *handle*)

This operation instructs the **DomainParticipant** (p. 44) **dp** to ignore a Publication identified by **handle**.

The handle can be discovered by accessing the built-in topic **DCPSPublication (p. 137)** via the appropriate built-in **DataReader** (p. 26).

There is no mechanism to reverse this operation.

Not Yet Supported This operation is not implemented yet.

2.6.2.30 ReturnCode_t DDS::DomainParticipant::ignore_subscription (InstanceHandle_t & *handle*)

This operation instructs the **DomainParticipant** (p. 44) **dp** to ignore a Subscription identified by **handle**.

The handle can be discovered by accessing the built-in topic **DCPSSubscription (p. 139)** via the appropriate built-in **DataReader** (p. 26).

There is no mechanism to reverse this operation.

Not Yet Supported This operation is not implemented yet.

2.6.2.31 ReturnCode_t DDS::DomainParticipant::ignore_topic (InstanceHandle_t & *handle*)

This operation instructs the **DomainParticipant** (p. 44) **dp** to ignore a **Topic** (p. 69) 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** (p. 26).

There is no mechanism to reverse this operation.

Not Yet Supported This operation is not implemented yet.

2.6.2.32 class TopicDescription* DDS::DomainParticipant::lookup_topicdescription (const char * *name*)

This operation returns an existing, locally-created **TopicDescription** (p. 72), named **name**.

If a **TopicDescription** (p. 72) named **name** does not exist, then this routine will return NULL.

2.6.2.33 ReturnCode_t DDS::DomainParticipant::register_type (TypeSupport * *ts*, const char * *type_name*)

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 provied type_name, then an error is returned and the previously registered TypeSupport is maintained.

2.6.2.34 ReturnCode_t DDS::DomainParticipant::set_default_publisher_qos (const PublisherQos & *qos*)

Sets the default **PublisherQos** (p. 78) held in the **DomainParticipant** (p. 44).

This default qos will be used during subsequent calls to **DomainParticipant::create_publisher()** (p. 47) if the special PUBLISHER_QOS_DEFAULT value is provided for **qos**.
This routine may fail if the provided **qos** argument is not internally consistent. In this case, INCONSISTENT_POLICY will be returned, and no changes will be made to the **DomainParticipant** (p. 44).

2.6.2.35 ReturnCode_t DDS::DomainParticipant::set_default_subscriber_qos (const SubscriberQos & *qos*)

Sets the default **SubscriberQos** (p. 79) held in the **DomainParticipant** (p. 44).

This default qos will be used during subsequent calls to **DomainParticipant::create_subscriber()** (p. 48) if the special SUBSCRIBER_QOS_DEFAULT value is provided for **qos**.
This routine may fail if the provided **qos** argument is not internally consistent. In this case, INCONSISTENT_POLICY will be returned, and no changes will be made to the **DomainParticipant** (p. 44).

2.6.2.36 ReturnCode_t DDS::DomainParticipant::set_default_topic_qos (const TopicQos & *qos*)

Sets the default **TopicQos** (p. 80) held in the **DomainParticipant** (p. 44).

This default qos will be used during subsequent calls to **DomainParticipant::create_topic()** (p. 48) [and related] if the special TOPIC_QOS_DEFAULT value is provided for **qos**.
This routine may fail if the provided **qos** argument is not internally consistent. In this case, INCONSISTENT_POLICY will be returned, and no changes will be made to the **DomainParticipant** (p. 44).

2.6.2.37 ReturnCode_t DDS::DomainParticipant::set_listener (DomainParticipantListener * *a_listener*, StatusMask *mask*)

This operation installs a **DomainParticipantListener** (p. 110) on the **DomainParticipant** (p. 44).

Only one listener may be attached to a **DomainParticipant** (p. 44) at a time. A call to **set_listener() (p. 53)** 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.

2.6.2.38 ReturnCode_t DDS::DomainParticipant::set_qos (const DomainParticipantQos & *qos*)

Sets the DomainParticipantQoS values.

These QoS values affect the behavior of the **DomainParticipant** (p. 44).

2.7 DDS::DomainParticipantFactory Class Reference

DomainParticipantFactory (p. 54) constructs DomainParticipants. The.

Public Member Functions

- **DomainParticipant** ∗ **create_participant** (**DomainId_t** domain_id, const **DomainParticipantQos** &qos, **DomainParticipantListener** ∗a_listener, **StatusMask** mask)

  *This operation creates a new **DomainParticipant** (p. 44) object.*
- **ReturnCode_t delete_participant** (**DomainParticipant** ∗a_participant)

  *Destroys the provided **DomainParticipant** (p. 44).*
- **DomainParticipant** ∗ **lookup_participant** (**DomainId_t** domain_id)

  *Returns a previously created **DomainParticipant** (p. 44) belonging to the specified **domain_id**.*
- **ReturnCode_t set_default_participant_qos** (const **DomainParticipantQos** &qos)

  *Sets the default **DomainParticipantQos** (p. 78) held in the factory.*
- **ReturnCode_t get_default_participant_qos** (**DomainParticipantQos** &qos)

  Provides access to the default Participant qos held in the factory.
- **ReturnCode_t set_qos** (const **DomainParticipantFactoryQos** &qos)

  *Sets the **DomainParticipantFactory** (p. 54) QoS values.*
- **ReturnCode_t get_qos** (**DomainParticipantFactoryQos** &qos)

  *Provides access to the QoS settings of the **DomainParticipantFactory** (p. 54).*

Static Public Member Functions

- static **DomainParticipantFactory** ∗ **get_instance** ()

2.7.1 Detailed Description

DomainParticipantFactory (p. 54) constructs DomainParticipants. The.

DomainParticipantFactory (p. 54) is used to configure, create and destroy **DomainParticipant** (p. 44) instances.

2.7.2 Member Function Documentation

2.7.2.1 DomainParticipant∗ DDS::DomainParticipantFactory::create_participant (DomainId_t *domain_id,* const DomainParticipantQos & *qos,* DomainParticipantListener ∗ *a_listener,* StatusMask *mask*)

This operation creates a new **DomainParticipant** (p. 44) 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** (p. 44) Quality of Service settings that should be used when creating the **DomainParticipant** (p. 44). It may be specified as **PARTICIPANT_QOS_DEFAULT** to instruct CoreDX DDS to use the default qos settings held in the **DomainParticipantFactory** (p. 54).
This routine will return **NULL** if it fails to create a **DomainParticipant** (p. 44).

2.7.2.2 ReturnCode_t DDS::DomainParticipantFactory::delete_participant (DomainParticipant * *a_participant*)

Destroys the provided **DomainParticipant** (p. 44).

This routine will fail if all Entities (Publishers, Subscribers, etc) created through the specified **DomainParticipant** (p. 44) have not yet been deleted. (In this case, RETCODE_PRECONDITION_NOT_MET will be returned.)

2.7.2.3 ReturnCode_t DDS::DomainParticipantFactory::get_default_participant_qos (DomainParticipantQos & *qos*)

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* DDS::DomainParticipantFactory::get_instance ( ) [static]

Get access to the singleton **DomainParticipantFactory** (p. 54).

2.7.2.5 DomainParticipant* DDS::DomainParticipantFactory::lookup_participant (DomainId_t *domain_id*)

Returns a previously created **DomainParticipant** (p. 44) belonging to the specified **domain_id**.

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

2.7.2.6 ReturnCode_t DDS::DomainParticipantFactory::set_default_participant_qos (const DomainParticipantQos & *qos*)

Sets the default **DomainParticipantQos** (p. 78) held in the factory.

This default qos will be used during subsequent calls to DomainParticipantFactory_create_participant() if the special PARTICIPANT_QOS_DEFAULT value is provided for **qos**.
This routine may fail if the provided **qos** argument is not internally consistent. In this case, INCONSISTENT_POLICY will be returned, and no changes will be made to the **DomainParticipantFactory** (p. 54).

2.7.2.7 ReturnCode_t DDS::DomainParticipantFactory::set_qos (const DomainParticipantFactoryQos & *qos*)

Sets the **DomainParticipantFactory** (p. 54) QoS values.

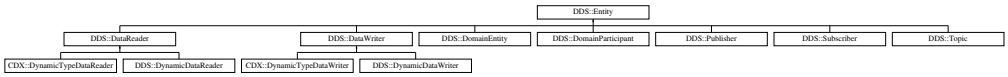
These QoS values affect the behavior of the factory.

This routine may fail if the provided **qos** argument is not internally consistent. In this case, INCONSISTENT_POLICY will be returned, and no changes will be made to the **DomainParticipantFactory** (p. 54).

2.8 DDS::Entity Class Reference

Base class for all DDS Entities.

Inheritance diagram for DDS::Entity:



Public Member Functions

- virtual **StatusCondition** ∗ **get_statuscondition** ()

  *Gets the **StatusCondition** (p. 64) associated with this **Entity** (p. 55). NOTE: The entity maintains ownership of the statuscondition, and it remains valid only as long as the **Entity** (p. 55) is valid. Destroying the **Entity** (p. 55) also destroys the status condition.*
- virtual **StatusMask get_status_changes** ()=0

  *Gets the current status changes from this **Entity** (p. 55).*
- virtual **ReturnCode_t enable** ()

  *Enable this **Entity** (p. 55).*
- virtual **InstanceHandle_t get_instance_handle** ()

  *Gets the handle that locally identifies this **Entity** (p. 55).*

2.8.1 Detailed Description

Base class for all DDS Entities.

2.8.2 Member Function Documentation

2.8.2.1 virtual ReturnCode_t DDS::Entity::enable ( ) [virtual]

Enable this **Entity** (p. 55).

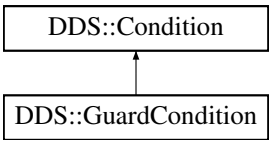
An **Entity** (p. 55) will begin to communicate only after it is enabled.

Reimplemented in **DDS::DataReader** (p. 29), **DDS::DataWriter** (p. 40), **DDS::Topic** (p. 70), **DDS::Subscriber** (p. 67), **DDS::Publisher** (p. 59), and **DDS::DomainParticipant** (p. 49).

2.9 DDS::GuardCondition Class Reference

A **GuardCondition** (p. 56) is a condition where the **trigger_value** is under application control.

Inheritance diagram for DDS::GuardCondition:



Public Member Functions

- bool **get_trigger_value** ()

  *This routine returns the current value of the **trigger_value** in **gc**.*
- **ReturnCode_t set_trigger_value** (bool v)

  *This routine set the current value of the **trigger_value** in **gc**.*

2.9.1 Detailed Description

A **GuardCondition** (p. 56) is a condition where the **trigger_value** is under application control.

2.9.2 Member Function Documentation

2.9.2.1 bool DDS::GuardCondition::get_trigger_value ( ) [virtual]

This routine returns the current value of the **trigger_value** in **gc**.

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

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

Reimplemented from **DDS::Condition** (p. 23).

2.9.2.2 ReturnCode_t DDS::GuardCondition::set_trigger_value (bool *v*)

This routine set the current value of the **trigger_value** in **gc**.

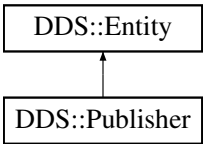
A non-zero **v** argument indicates that the **trigger_value** is TRUE.

A zero **v** argument indicates that the **trigger_value** is FALSE. Setting the trigger value to a non-zero state will cause any threads that are waiting on a **WaitSet** (p. 73) with this **GuardCondition** (p. 56) attached to unblock. [In other words, the WaitSet_wait() routine will return if the **WaitSet** (p. 73) has this **GuardCondition** (p. 56) attached to it.]

2.10 DDS::Publisher Class Reference

The **Publisher** (p. 57) configures, creates, manages and destroys DataWriters.

Inheritance diagram for DDS::Publisher:



Public Member Functions

- **ReturnCode_t enable** ()

  *Enables the **Publisher** (p. 57).*
- **InstanceHandle_t get_instance_handle** ()

  *This operation returns the InstanceHandle_t that identifies the **Publisher** (p. 57).*
- **StatusMask get_status_changes** ()

  *This returns the list of **triggered** communication statuses in the **Publisher** (p. 57).*
- class **DomainParticipant** ∗ **get_participant** ()

  *This operation returns the **DomainParticipant** (p. 44) this **Publisher** (p. 57) belongs to.*
- class **DataWriter** ∗ **create_datawriter** (**Topic** ∗a_topic, const **DataWriterQos** &qos, **DataWriterListener** ∗a_listener, **StatusMask** mask)

  *This operation creates a **DataWriter** (p. 37).*

- **ReturnCode_t delete_datawriter** (**DataWriter** ∗a_datawriter)

  *This operation deletes a **DataWriter** (p. 37).*
- class **DataWriter** ∗ **lookup_datawriter** (const char ∗topic_name)

  *This operation retrieves a previously-created **DataWriter** (p. 37) contained in the **Publisher** (p. 57), attached to a **Topic** (p. 69) named **topic_name**.*
- **ReturnCode_t delete_contained_entities** ()

  *This operation deletes all the DataWriters created by means of the **Publisher::create_datawriter()** (p. 59) operation on the **Publisher** (p. 57) **p**.*
- **ReturnCode_t set_qos** (const **PublisherQos** &qos)

  *Sets the **PublisherQos** (p. 78) values.*
- **ReturnCode_t get_qos** (**PublisherQos** &qos)

  *Returns the current **PublisherQos** (p. 78) settings held in the **Publisher** (p. 57) **p**.*
- **ReturnCode_t set_listener** (**PublisherListener** ∗a_listener, **StatusMask** mask)

  *Installs a **PublisherListener** (p. 110) on **Publisher** (p. 57) **p**.*
- class **PublisherListener** ∗ **get_listener** ()

  *This operation returns the currently installed **PublisherListener** (p. 110).*
- **ReturnCode_t suspend_publications** ()
- **ReturnCode_t resume_publications** ()
- **ReturnCode_t begin_coherent_changes** ()
- **ReturnCode_t end_coherent_changes** ()
- **ReturnCode_t wait_for_acknowledgments** (const **Duration_t** &max_wait)

  Block until all writers contained by this publisher have received acknowledgements.
- **ReturnCode_t set_default_datawriter_qos** (const **DataWriterQos** &qos)

  *Sets the default **DataWriterQos** (p. 76) held in the **Publisher** (p. 57).*
- **ReturnCode_t get_default_datawriter_qos** (**DataWriterQos** &qos)

  *Provides access to the default **DataWriterQos** (p. 76) settings held in the **Publisher** (p. 57) **p**.*
- **ReturnCode_t copy_from_topic_qos** (**DataWriterQos** &a_datawriter_qos, const **TopicQos** &a_topic_qos)

  *This operation copies the QoS settings in **a_topic_qos** to the corresponding settings in **a_datawriter_qos**.*

2.10.1 Detailed Description

The **Publisher** (p. 57) configures, creates, manages and destroys DataWriters.

2.10.2 Member Function Documentation

2.10.2.1 ReturnCode_t DDS::Publisher::begin_coherent_changes ( )

Not Yet Supported This operation is not yet implemented.

2.10.2.2 ReturnCode_t DDS::Publisher::copy_from_topic_qos (DataWriterQos & *a_datawriter_qos,* const TopicQos & *a_topic_qos*)

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.

2.10.2.3 class DataWriter* DDS::Publisher::create_datawriter (Topic * *a_topic,* const DataWriterQos & *qos,* DataWriterListener * *a_listener,* StatusMask *mask*)

This operation creates a **DataWriter** (p. 37).

The created **DataWriter** (p. 37) is contained within the **Publisher** (p. 57) **p**. It is associated with the **Topic** (p. 69), **ContentFilteredTopic** (p. 24), or MultiTopic indicated by **a_topic**, and has the **DataWriterQos** (p. 76) indicated by **qos**. The **qos** argument may be passed DATAWRITER_QOS_DEFAULT, which indicates that the **Publisher** (p. 57) should use its currently configured default data writer QoS values. The **DataWriterListener** (p. 109) **a_listener**, is installed at creation time.

The created **DataWriter** (p. 37) (if not NULL) must be destroyed by a call to **Publisher::delete_datawriter()** (p. 59).

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

2.10.2.4 ReturnCode_t DDS::Publisher::delete_contained_entities ( )

This operation deletes all the DataWriters created by means of the **Publisher::create_datawriter()** (p. 59) operation on the **Publisher** (p. 57) **p**.

This routine will recursively call the corresponding **delete_contained_entities()** (p. 59) operation on each of the contained **DataWriter** (p. 37) objects. After successful execution, the application may delete the **Publisher** (p. 57) by calling **DomainParticipant::delete_publisher()** (p. 49).

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

2.10.2.5 ReturnCode_t DDS::Publisher::delete_datawriter (DataWriter * *a_datawriter*)

This operation deletes a **DataWriter** (p. 37).

If the provided **DataWriter** (p. 37) **a_datawriter** was not created by **Publisher** (p. 57) **p**, the routine will fail and will return RETCODE_PRECONDITION_NOT_MET.

2.10.2.6 ReturnCode_t DDS::Publisher::enable ( ) [virtual]

Enables the **Publisher** (p. 57).

A **Publisher** (p. 57) is created either enabled or not based on the **DomainParticipantQos** (p. 78) setting **entity_factory**. When a **Publisher** (p. 57) is not enabled, only the following sub-set of all **Publisher** (p. 57) operations are legal:

- operations to get and set QoS policies,
- factory operations (create, delete),
- **get_statuscondition()** (p. 56),
- **get_status_changes()** (p. 60),
- lookup operations

Any other operation may return the NOT_ENABLED error. **Publisher::enable()** (p. 59) may be called on an already enabled **Subscriber** (p. 65) [it will have no effect].

Reimplemented from **DDS::Entity** (p. 56).

2.10.2.7 ReturnCode_t DDS::Publisher::end_coherent_changes ( )

Not Yet Supported This operation is not yet implemented.

2.10.2.8 ReturnCode_t DDS::Publisher::get_default_datawriter_qos (DataWriterQos & *qos*)

Provides access to the default **DataWriterQos** (p. 76) settings held in the **Publisher** (p. 57) **p**.

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

2.10.2.9 class PublisherListener* DDS::Publisher::get_listener ( )

This operation returns the currently installed **PublisherListener** (p. 110).

Note

Because the infrastructure makes a copy of the listener provided in **Publisher::set_listener()** (p. 61), 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.

2.10.2.10 ReturnCode_t DDS::Publisher::get_qos (PublisherQos & *qos*)

Returns the current **PublisherQos** (p. 78) settings held in the **Publisher** (p. 57) **p**.

The **qos** parameter is populated with a copy of the current **Publisher** (p. 57) QoS properties.

Note

The qos structure may contain sequences or strings that are populated with dynamic memory. The caller is responsible for freeing the dynamic memory of these items.
For example, the sequence 'qos->group_data' may have dynamically allocated memory assigned. This can be released by a call to **seq_clear(&qos->group_data)**.

2.10.2.11 StatusMask DDS::Publisher::get_status_changes ( ) [virtual]

This returns the list of **triggered** communication statuses in the **Publisher** (p. 57).

If the **Publisher** (p. 57) is not enabled, all statuses will be **untriggered**. The list returned by **get_status_changes** may be empty. The list of statuses returned by **get_status_changes** operation contains statuses that are triggered on the **Publisher** (p. 57) itself and does not include statuses from contained **DataWriter (p. 37)** objects.

Implements **DDS::Entity** (p. 56).

2.10.2.12 class DataWriter* DDS::Publisher::lookup_datawriter (const char * *topic_name*)

This operation retrieves a previously-created **DataWriter** (p. 37) contained in the **Publisher** (p. 57), attached to a **Topic** (p. 69) named **topic_name**.

If multiple DataWriters are found, one of them will be returned. If no matching **DataWriter** (p. 37) is found, this routine will return **NULL**.

2.10.2.13 ReturnCode_t DDS::Publisher::resume_publications ( )

Not Yet Supported This operation is not yet implemented.

2.10.2.14 ReturnCode_t DDS::Publisher::set_default_datawriter_qos (const DataWriterQos & *qos*)

Sets the default **DataWriterQos** (p. 76) held in the **Publisher** (p. 57).

This default qos will be used during subsequent calls to **Publisher::create_datawriter()** (p. 59) if the special DATAWRITER_QOS_DEFAULT value is provided for **qos**.
This routine may fail if the provided **qos** argument is not internally consistent. In this case, INCONSISTENT_POLICY will be returned, and no changes will be made to the **Publisher** (p. 57).

2.10.2.15 ReturnCode_t DDS::Publisher::set_listener (PublisherListener * *a_listener,* StatusMask *mask*)

Installs a **PublisherListener** (p. 110) on **Publisher** (p. 57) **p**.

Only one listener may be attached to a **Publisher** (p. 57) at a time. A call to **set_listener() (p. 61)** 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.

2.10.2.16 ReturnCode_t DDS::Publisher::set_qos (const PublisherQos & *qos*)

Sets the **PublisherQos** (p. 78) values.

These QoS values affect the behavior of the **Publisher** (p. 57).

2.10.2.17 ReturnCode_t DDS::Publisher::suspend_publications ( )

Not Yet Supported This operation is not yet implemented.

2.10.2.18 ReturnCode_t DDS::Publisher::wait_for_acknowledgments (const Duration_t & *max_wait*)

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.

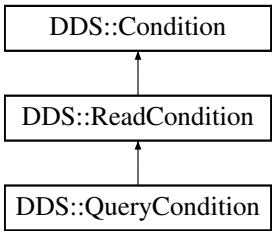
Return values

RETCODE_TIME_OUT	returned if 'max_wait' passes before all acks are received
RETCODE_OK	returned if all acks have been received before 'max_wait'

2.11 DDS::QueryCondition Class Reference

A **QueryCondition** (p. 61) is a specialized **ReadCondition** (p. 63) which includes a filter.

Inheritance diagram for DDS::QueryCondition:



Public Member Functions

- const char ∗ **get_query_expression** (void)

  *This routine returns the SQL query expression of this **QueryCondition** (p. 61).*
- **ReturnCode_t get_query_parameters** (**StringSeq** &seq)

  *This routine returns the parameters to the SQL query expression of this **QueryCondition** (p. 61).*
- **ReturnCode_t set_query_parameters** (const **StringSeq** &seq)

  *This routine sets the parameters to the SQL query expression of this **QueryCondition** (p. 61).*

2.11.1 Detailed Description

A **QueryCondition** (p. 61) is a specialized **ReadCondition** (p. 63) which includes a filter.

The **trigger_value** is driven by the data available, after applying the filter, in the associated **DataReader** (p. 26).

Not Yet Supported CoreDX DDS does not yet implement QueryConditions as a trigger for a **WaitSet** (p. 73).

2.11.2 Member Function Documentation

2.11.2.1 const char∗ DDS::QueryCondition::get_query_expression (void)

This routine returns the SQL query expression of this **QueryCondition** (p. 61).

The query expression is an SQL syntax conditional expression that is provided when the **QueryCondition** (p. 61) is created.

2.11.2.2 ReturnCode_t DDS::QueryCondition::get_query_parameters (StringSeq & *seq*)

This routine returns the parameters to the SQL query expression of this **QueryCondition** (p. 61).

The query expression is an SQL syntax conditional expression that is provided when the **QueryCondition** (p. 61) 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()

2.11.2.3 ReturnCode_t DDS::QueryCondition::set_query_parameters (const StringSeq & *seq*)

This routine sets the parameters to the SQL query expression of this **QueryCondition** (p. 61).

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.

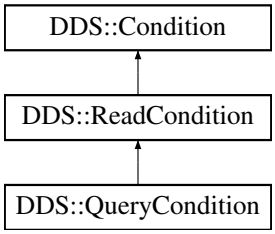
See also

DataReader_create_querycondition()

2.12 DDS::ReadCondition Class Reference

A **ReadCondition** (p. 63) is a specialized **Condition** (p. 23) associated with a **DataReader** (p. 26).

Inheritance diagram for DDS::ReadCondition:



Public Member Functions

- bool **get_trigger_value** ()

  *This routine returns the current value of the **trigger_value** in **rc**.*
- **DataReader** ∗ **get_datareader** ()

  *This routine returns the single **DataReader** (p. 26) associated with this **ReadCondition** (p. 63).*
- SampleStateKind **get_sample_state_mask** ()

  *This routine returns the current value of the sample_state in this **ReadCondition** (p. 63).*
- ViewStateKind **get_view_state_mask** ()

  *This routine returns the current value of the view_state in this **ReadCondition** (p. 63).*
- InstanceStateKind **get_instance_state_mask** ()

  *This routine returns the current value of the instance_state in this **ReadCondition** (p. 63).*

2.12.1 Detailed Description

A **ReadCondition** (p. 63) is a specialized **Condition** (p. 23) associated with a **DataReader** (p. 26).

The **trigger_value** is driven by the data available in the associated **DataReader** (p. 26). A **ReadCondition** (p. 63) is obtained by calling the DataReader_create_readcondition() function. When the **ReadCondition** (p. 63) is no longer needed, it should be destroyed by a call to DataReader_delete_readcondition().

2.12.2 Member Function Documentation

2.12.2.1 bool DDS::ReadCondition::get_trigger_value ( ) [virtual]

This routine returns the current value of the **trigger_value** in **rc**.

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

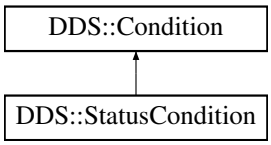
A zero return value indicates that the **trigger_value** is FALSE.

Reimplemented from **DDS::Condition** (p. 23).

2.13 DDS::StatusCondition Class Reference

A **StatusCondition** (p. 64) is a condition associated with an **Entity** (p. 55).

Inheritance diagram for DDS::StatusCondition:



Public Member Functions

- bool **get_trigger_value** ()

  *This routine returns the current value of the **trigger_value** in **gc**.*
- **ReturnCode_t set_enabled_statuses** (**StatusMask** mask)

  *This routine sets the statuses which are enabled in this **StatusCondition** (p. 64).*
- **StatusMask get_enabled_statuses** ()

  *This routine returns the statuses which are enabled in this **StatusCondition** (p. 64).*
- **Entity** ∗ **get_entity** ()

  *This routine returns the single entity associated with this **StatusCondition** (p. 64).*

2.13.1 Detailed Description

A **StatusCondition** (p. 64) is a condition associated with an **Entity** (p. 55).

The **trigger_value** is driven by the communication status of the associated **Entity** (p. 55).

2.13.2 Member Function Documentation

2.13.2.1 StatusMask DDS::StatusCondition::get_enabled_statuses ( )

This routine returns the statuses which are enabled in this **StatusCondition** (p. 64).

The statuses are returned as a bitmask.

2.13.2.2 bool DDS::StatusCondition::get_trigger_value ( ) [virtual]

This routine returns the current value of the **trigger_value** in **gc**.

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

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

Reimplemented from **DDS::Condition** (p. 23).

2.13.2.3 ReturnCode_t DDS::StatusCondition::set_enabled_statuses (StatusMask *mask*)

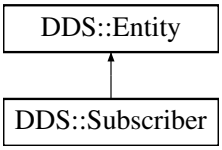
This routine sets the statuses which are enabled in this **StatusCondition** (p. 64).

The statuses are provided as a bitmask.

2.14 DDS::Subscriber Class Reference

The **Subscriber** (p. 65) configures, creates, manages and destroys DataReaders.

Inheritance diagram for DDS::Subscriber:



Public Member Functions

- **ReturnCode_t enable** ()

  *Enables the **Subscriber** (p. 65).*
- **InstanceHandle_t get_instance_handle** ()

  *This operation returns the InstanceHandle_t that identifies the **Subscriber** (p. 65).*
- **StatusMask get_status_changes** ()

  *This returns the list of **triggered** communication statuses in the **Subscriber** (p. 65).*
- **DataReader** ∗ **create_datareader** (**TopicDescription** ∗a_topic, const **DataReaderQos** &qos, **DataReaderListener** ∗a_listener, **StatusMask** mask)

  *This operation creates a **DataReader** (p. 26).*
- **ReturnCode_t delete_datareader** (class **DataReader** ∗a_datareader)

  *This operation deletes a **DataReader** (p. 26).*
- **ReturnCode_t delete_contained_entities** ()

  *This operation deletes all the DataReaders created by means of the **Subscriber::create_datareader()** (p. 67) operation on the **Subscriber** (p. 65) **s**.*
- **DataReader** ∗ **lookup_datareader** (const char ∗topic_name)

  *This operation retrieves a previously-created **DataReader** (p. 26) contained in the **Subscriber** (p. 65), attached to a **Topic** (p. 69) named **topic_name**.*
- **ReturnCode_t get_datareaders** (**DataReaderSeq** ∗readers, SampleStateMask sample_states, ViewStateMask view_states, InstanceStateMask instance_states)

  *This operation allows the application to access **DataReader** (p. 26) objects that contain samples with the specified **sample_states**, **view_states**, and **instance_states**.*

- **DomainParticipant** ∗ **get_participant** ()

  *This operation returns the **DomainParticipant** (p. 44) this **Subscriber** (p. 65) belongs to.*

- **ReturnCode_t set_qos** (const **SubscriberQos** &qos)

  *Sets the **SubscriberQos** (p. 79) values.*

- **ReturnCode_t get_qos** (**SubscriberQos** &qos)

  *Returns the current **SubscriberQos** (p. 79) settings held in the **Subscriber** (p. 65) **s**.*

- **ReturnCode_t set_listener** (**SubscriberListener** ∗a_listener, **StatusMask** mask)

  *Installs a **SubscriberListener** (p. 111) on **Subscriber** (p. 65) **s**.*

- **SubscriberListener** ∗ **get_listener** ()

  *This operation returns the currently installed **SubscriberListener** (p. 111).*

- **ReturnCode_t begin_access** ()

  *This operation indicates that the application is about to access the data samples in any of the **DataReader** (p. 26) objects contained in the **Subscriber** (p. 65) **s**.*

- **ReturnCode_t end_access** ()

  *This operation closes a corresponding **Subscriber::begin_access()** (p. 66).*

- **ReturnCode_t set_default_datareader_qos** (const **DataReaderQos** &qos)

  *Sets the default **DataReaderQos** (p. 75) held in the **Subscriber** (p. 65).*

- **ReturnCode_t get_default_datareader_qos** (**DataReaderQos** &qos)

  *Provides access to the default **DataReaderQos** (p. 75) settings held in the **Subscriber** (p. 65) **s**.*

- **ReturnCode_t copy_from_topic_qos** (**DataReaderQos** &a_datareader_qos, const **TopicQos** &a_topic_qos)

  *This operation copies the QoS settings in **a_topic_qos** to the corresponding settings in **a_datareader_qos**.*

2.14.1 Detailed Description

The **Subscriber** (p. 65) configures, creates, manages and destroys DataReaders.

2.14.2 Member Function Documentation

2.14.2.1 ReturnCode_t DDS::Subscriber::begin_access ( )

This operation indicates that the application is about to access the data samples in any of the **DataReader** (p. 26) objects contained in the **Subscriber** (p. 65) **s**.

The application is expected to use this operation only if PRESENTATION QosPolicy of the **Subscriber** (p. 65) has the access_scope set to GROUP. [Otherwise this routine has no effect.]

2.14.2.2 ReturnCode_t DDS::Subscriber::copy_from_topic_qos (DataReaderQos & *a_datareader_qos*, const TopicQos & *a_topic_qos*)

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.

2.14.2.3 DataReader* DDS::Subscriber::create_datareader (TopicDescription * *a_topic,* const DataReaderQos & *qos,* DataReaderListener * *a_listener,* StatusMask *mask*)

This operation creates a **DataReader** (p. 26).

The created **DataReader** (p. 26) is contained within the **Subscriber** (p. 65) **s**. It is associated with the **Topic** (p. 69), **ContentFilteredTopic** (p. 24), or MultiTopic indicated by **a_topic**, and has the **DataReaderQos** (p. 75) indicated by **qos**. The **qos** argument may be passed DATAREADER_QOS_DEFAULT, which indicates that the **Subscriber** (p. 65) should use its currently configured default data reader QoS values. The **DataReaderListener** (p. 107) **a_listener**, is installed at creation time.

The created **DataReader** (p. 26) (if not NULL) must be destroyed by a call to **Subscriber::delete_datareader()** (p. 67).

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

2.14.2.4 ReturnCode_t DDS::Subscriber::delete_contained_entities ( )

This operation deletes all the DataReaders created by means of the **Subscriber::create_datareader()** (p. 67) operation on the **Subscriber** (p. 65) **s**.

This routine will recursively call the corresponding **delete_contained_entities()** (p. 67) operation on each of the contained **DataReader** (p. 26) objects. If successful, this operation will recursively delete all objects contained with this **Subscriber** (p. 65). After successful execution, the application may delete the **Subscriber** (p. 65) by calling **DomainParticipant::delete_subscriber()** (p. 49).

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

2.14.2.5 ReturnCode_t DDS::Subscriber::delete_datareader (class DataReader * *a_datareader*)

This operation deletes a **DataReader** (p. 26).

If the provided **DataReader** (p. 26) **a_datareader** was not created by **Subscriber** (p. 65) **s**, the routine will fail and will return RETCODE_PRECONDITION_NOT_MET. If the indicated **DataReader** (p. 26) has any outstanding **ReadCondition** (p. 63) or **QueryCondition** (p. 61) objects the routine will fail and will return RETCODE_PRECONDITION_NOT_MET. If the indicated **DataReader** (p. 26) has any outstanding 'loans' (from read() or take() operations), the routine will fail and will return RETCODE_PRECONDITION_NOT_MET.

2.14.2.6 ReturnCode_t DDS::Subscriber::enable ( ) [virtual]

Enables the **Subscriber** (p. 65).

A **Subscriber** (p. 65) is created either enabled or not based on the **DomainParticipantQos** (p. 78) setting **entity_factory**. When a **Subscriber** (p. 65) is not enabled, only the following sub-set of all **Subscriber** (p. 65) operations are legal:

- operations to get and set QoS policies,
- factory operations (create, delete),
- **get_statuscondition()** (p. 56),
- **get_status_changes()** (p. 69),
- lookup operations

Any other **Subscriber** (p. 65) operation may return the NOT_ENABLED error. **Subscriber::enable()** (p. 67) may be called on an already enabled **Subscriber** (p. 65) [it will have no effect].

Reimplemented from **DDS::Entity** (p. 56).

2.14.2.7 ReturnCode_t DDS::Subscriber::get_datareaders (DataReaderSeq * *readers,* SampleStateMask *sample_states,* ViewStateMask *view_states,* InstanceStateMask *instance_states*)

This operation allows the application to access **DataReader** (p. 26) objects that contain samples with the specified **sample_states**, **view_states**, and **instance_states**.

If the PRESENTATION QosPolicy of the **Subscriber** (p. 65) to which the **DataReader** (p. 26) 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** (p. 26) objects may either be a set (containing each **DataReader** (p. 26) at most once in no specified order), or a list (containing each **DataReader** (p. 26) 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.

2.14.2.8 ReturnCode_t DDS::Subscriber::get_default_datareader_qos (DataReaderQos & *qos*)

Provides access to the default **DataReaderQos** (p. 75) settings held in the **Subscriber** (p. 65) **s**.

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

2.14.2.9 SubscriberListener* DDS::Subscriber::get_listener ( )

This operation returns the currently installed **SubscriberListener** (p. 111).

Note

Because the infrastructure makes a copy of the listener provided in **Subscriber::set_listener()** (p. 69), 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.

2.14.2.10 ReturnCode_t DDS::Subscriber::get_qos (SubscriberQos & *qos*)

Returns the current **SubscriberQos** (p. 79) settings held in the **Subscriber** (p. 65) **s**.

The **qos** parameter is populated with a copy of the current **Subscriber** (p. 65) QoS properties.

Note

The qos structure may contain sequences or strings that are populated with dynamic memory. The caller is responsible for freeing the dynamic memory of these items.
For example, the sequence 'qos->group_data' may have dynamically allocated memory assigned. This can be released by a call to **seq_clear(&qos->group_data)**.

2.14.2.11 StatusMask DDS::Subscriber::get_status_changes ( ) [virtual]

This returns the list of **triggered** communication statuses in the **Subscriber** (p. 65).

If the **Subscriber** (p. 65) is not enabled, all statuses will be **untriggered**. The list returned by **get_status_changes** may be empty. The list of statuses returned by **get_status_changes** operation contains statuses that are triggered on the **Subscriber** (p. 65) itself and does not include statuses from contained **DataReader (p. 26)** objects.

Implements **DDS::Entity** (p. 56).

2.14.2.12 DataReader* DDS::Subscriber::lookup_datareader (const char * *topic_name*)

This operation retrieves a previously-created **DataReader** (p. 26) contained in the **Subscriber** (p. 65), attached to a **Topic** (p. 69) named **topic_name**.

If multiple DataReaders are found, one of them will be returned. If no matching **DataReader** (p. 26) is found, this routine will return **NULL**.

This routine is useful to obtain access to a particular built-in **DataReader** (p. 26).

2.14.2.13 ReturnCode_t DDS::Subscriber::set_default_datareader_qos (const DataReaderQos & *qos*)

Sets the default **DataReaderQos** (p. 75) held in the **Subscriber** (p. 65).

This default qos will be used during subsequent calls to **Subscriber::create_datareader()** (p. 67) if the special DATAREADER_QOS_DEFAULT value is provided for **qos**.
This routine may fail if the provided **qos** argument is not internally consistent. In this case, INCONSISTENT_POLICY will be returned, and no changes will be made to the **Subscriber** (p. 65).

2.14.2.14 ReturnCode_t DDS::Subscriber::set_listener (SubscriberListener * *a_listener,* StatusMask *mask*)

Installs a **SubscriberListener** (p. 111) on **Subscriber** (p. 65) **s**.

Only one listener may be attached to a **Subscriber** (p. 65) at a time. A call to **set_listener() (p. 69)** 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.

2.14.2.15 ReturnCode_t DDS::Subscriber::set_qos (const SubscriberQos & *qos*)

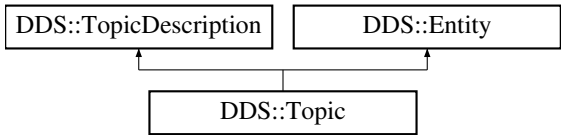
Sets the **SubscriberQos** (p. 79) values.

These QoS values affect the behavior of the **Subscriber** (p. 65).

2.15 DDS::Topic Class Reference

Topic (p. 69) is the basic description of data to be published or subscribed.

Inheritance diagram for DDS::Topic:



Public Member Functions

- const char ∗ **get_type_name** ()

  *This operation returns **type_name** of the provided **Topic** (p. 69).*
- const char ∗ **get_name** ()

  *This operation returns **topic name** of the provided **Topic** (p. 69).*
- **DomainParticipant** ∗ **get_participant** ()

  *This operation returns **DomainParticipant** (p. 44) that owns the **Topic** (p. 69).*
- **StatusMask get_status_changes** ()

  *This returns the list of **triggered** communication statuses in the **Topic** (p. 69).*
- **ReturnCode_t enable** ()

  *Enables the **Topic** (p. 69).*
- **ReturnCode_t set_qos** (const **TopicQos** &qos)

  *Sets the **TopicQos** (p. 80) values.*
- **ReturnCode_t get_qos** (**TopicQos** &qos)

  *Returns the current **TopicQos** (p. 80) settings held in the **Topic** (p. 69) **t**.*
- **ReturnCode_t set_listener** (**TopicListener** ∗a_listener, **StatusMask** mask)

  *Installs a **TopicListener** (p. 112) on **Topic** (p. 69) **t**.*
- **TopicListener** ∗ **get_listener** ()

  *This operation returns the currently installed **TopicListener** (p. 112).*
- **ReturnCode_t get_inconsistent_topic_status** (InconsistentTopicStatus ∗a_status)

  *Provides access to the InconsistentTopicStatus of the **Topic** (p. 69).*

2.15.1 Detailed Description

Topic (p. 69) is the basic description of data to be published or subscribed.

A topic is identified by a **name** and a **type**. A **Topic** (p. 69) is created by calling **DomainParticipant::create_topic()** (p. 48). Prior to creating a **Topic** (p. 69), the associated data type must be registered with the **DomainParticipant** (p. 44) via a call to the TypeSupport::register_type() function. [The register_type() function is auto-generated 'type-specific' code.]

2.15.2 Member Function Documentation

2.15.2.1 ReturnCode_t DDS::Topic::enable ( ) [virtual]

Enables the **Topic** (p. 69).

A **Topic** (p. 69) is created either enabled or not based on the **DomainParticipantQos** (p. 78) setting **entity_factory**. When a **Topic** (p. 69) is not enabled, only the following sub-set of all **Topic** (p. 69) operations are legal:

- operations to get and set QoS policies,

- **get_name()** (p. 70), **get_type_name()** (p. 70), **get_participant()** (p. 70)
- **get_statuscondition()** (p. 56),
- **get_status_changes()** (p. 71),

Any other operation may return the NOT_ENABLED error. **Topic::enable()** (p. 70) may be called on an already enabled **Topic** (p. 69) [it will have no effect].

Reimplemented from **DDS::Entity** (p. 56).

2.15.2.2 ReturnCode_t DDS::Topic::get_inconsistent_topic_status (InconsistentTopicStatus * *a_status*)

Provides access to the InconsistentTopicStatus of the **Topic** (p. 69).

As a side-effect, this routine will reset the **total_count_change** status field to zero.

2.15.2.3 TopicListener* DDS::Topic::get_listener ( )

This operation returns the currently installed **TopicListener** (p. 112).

Note

Because the infrastructure makes a copy of the listener provided in **Topic::set_listener()** (p. 71), 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.

2.15.2.4 ReturnCode_t DDS::Topic::get_qos (TopicQos & *qos*)

Returns the current **TopicQos** (p. 80) settings held in the **Topic** (p. 69) **t**.

This routine copies the **Topic** (p. 69) QoS properites into **qos**.

Note

The qos structure may contain sequences or strings that are populated with dynamic memory. The caller is responsible for freeing the dynamic memory of these items.
For example, the sequence 'qos->topic_data' may have dynamically allocated memory assigned. This can be released by a call to **seq_clear(&qos->topic_data)**.

2.15.2.5 StatusMask DDS::Topic::get_status_changes ( ) [virtual]

This returns the list of **triggered** communication statuses in the **Topic** (p. 69).

If the **Topic** (p. 69) is not enabled, all statuses will be **untriggered**.

Implements **DDS::Entity** (p. 56).

2.15.2.6 ReturnCode_t DDS::Topic::set_listener (TopicListener * *a_listener,* StatusMask *mask*)

Installs a **TopicListener** (p. 112) on **Topic** (p. 69) **t**.

Only one listener may be attached to a **Topic** (p. 69) at a time. A call to **set_listener() (p. 71)** 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.

2.15.2.7 ReturnCode_t DDS::Topic::set_qos (const TopicQos & *qos*)

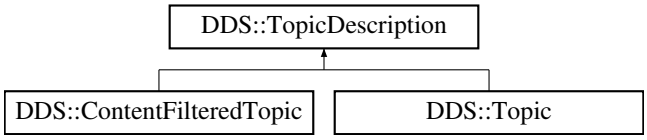
Sets the **TopicQos** (p. 80) values.

These QoS values affect the behavior of the **Topic** (p. 69).

2.16 DDS::TopicDescription Class Reference

TopicDescription (p. 72) is an abstract 'class' that provides the foundation for **Topic** (p. 69), **ContentFilteredTopic** (p. 24), and MultiTopic.

Inheritance diagram for DDS::TopicDescription:



Public Member Functions

- virtual **DomainParticipant** ∗ **get_participant** ()

  *This operation returns **DomainParticipant** (p. 44) that owns the **TopicDescription** (p. 72).*
- virtual const char ∗ **get_type_name** ()

  *This operation returns **type_name** of the provided **TopicDescription** (p. 72).*
- virtual const char ∗ **get_name** ()

  *This operation returns **topic name** of the provided **TopicDescription** (p. 72).*

2.16.1 Detailed Description

TopicDescription (p. 72) is an abstract 'class' that provides the foundation for **Topic** (p. 69), **ContentFilteredTopic** (p. 24), and MultiTopic.

2.16.2 Member Function Documentation

2.16.2.1 virtual const char∗ DDS::TopicDescription::get_name ( ) [virtual]

This operation returns **topic name** of the provided **TopicDescription** (p. 72).

The returned string is owned by the **TopicDescription (p. 72)**, do not free it.

Reimplemented in **DDS::ContentFilteredTopic** (p. 25), and **DDS::Topic** (p. 70).

2.16.2.2 virtual const char* DDS::TopicDescription::get_type_name ( ) [virtual]

This operation returns **type_name** of the provided **TopicDescription** (p. 72).

The returned string is owned by the **TopicDescription (p. 72)**, do not free it.

Reimplemented in **DDS::ContentFilteredTopic** (p. 25), and **DDS::Topic** (p. 70).

2.17 DDS::WaitSet Class Reference

A **WaitSet** (p. 73) maintains a set of **Condition** (p. 23) objects and allows the application to wait until one or more of them have a **trigger_value** of TRUE.

Public Member Functions

- **ReturnCode_t attach_condition** (**Condition** &a_condition)

  *Adds the condition **c** to the **WaitSet** (p. 73) **ws**.*
- **ReturnCode_t detach_condition** (**Condition** &a_condition)

  *Removes a condition **c** from **WaitSet** (p. 73) **ws**.*
- **ReturnCode_t wait** (**ConditionSeq** &active_conditions, const **Duration_t** &timeout)

  *Causes the controlling thread to block until an attached condition is triggered or **timeout** elapses.*
- **ReturnCode_t get_conditions** (**ConditionSeq** &attached_conditions)

  Retrieves the current list of attached conditions.

2.17.1 Detailed Description

A **WaitSet** (p. 73) maintains a set of **Condition** (p. 23) 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** (p. 73). A **WaitSet** (p. 73) is created by a call to WaitSet__alloc(), and destroyed with a call to WaitSet__free().

2.17.2 Member Function Documentation

2.17.2.1 ReturnCode_t DDS::WaitSet::attach_condition (Condition & *a_condition*)

Adds the condition **c** to the **WaitSet** (p. 73) **ws**.

If another thread is currently 'waiting' on the **WaitSet** (p. 73), this newly added condition will unblock that thread if its **trigger_value** is TRUE.

2.17.2.2 ReturnCode_t DDS::WaitSet::detach_condition (Condition & *a_condition*)

Removes a condition **c** from **WaitSet** (p. 73) **ws**.

If the condition is not attached to the **WaitSet** (p. 73), the error RETCODE_PRECONDITION_NOT_MET will be returned.

2.17.2.3 ReturnCode_t DDS::WaitSet::get_conditions (ConditionSeq & *attached_conditions*)

Retrieves the current list of attached conditions.

Populates the **attached_conditions** sequence. The application is responsible for freeing the memory contained in the sequence.

2.17.2.4 ReturnCode_t DDS::WaitSet::wait (ConditionSeq & *active_conditions,* const Duration_t & *timeout*)

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

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

Note

The application is responsible for clearing the memory contained in **active_conditions** upon return.

Chapter 3

Entity QoS Collections

3.1 DDS::DataReaderQos Class Reference

Structure that holds **DataReader** (p. 26) Quality of Service policies.

Public Attributes

- **DurabilityQosPolicy durability**

  *The durability policy requested by the **DataReader** (p. 26).*
- **DeadlineQosPolicy deadline**

  The requested update frequency for data instances.
- **LatencyBudgetQosPolicy latency_budget**

  *The latency requested by the **DataReader** (p. 26).*
- **LivelinessQosPolicy liveliness**

  *The liveliness mechanism requested by the **DataReader** (p. 26).*
- **ReliabilityQosPolicy reliability**

  *The transport reliability requested by the **DataReader** (p. 26).*
- **DestinationOrderQosPolicy destination_order**

  *The destination order logic requested by the **DataReader** (p. 26).*
- **HistoryQosPolicy history**

  *The data history requested by the **DataReader** (p. 26).*
- **ResourceLimitsQosPolicy resource_limits**

  *The resource limits set on the **DataReader** (p. 26).*
- **UserDataQosPolicy user_data**

  *A sequence of octets associated with the **DataReader** (p. 26).*
- **OwnershipQosPolicy ownership**

  *The type of 'ownership' offered by the **DataReader** (p. 26).*
- **TimeBasedFilterQosPolicy time_based_filter**

  *The maximum update frequency required/desired by the **DataReader** (p. 26).*
- **ReaderDataLifecycleQosPolicy reader_data_lifecycle**

  *Controls the auto-purge behavior of the **DataReader** (p. 26).*
- **DataRepresentationQosPolicy representation**

  *Informs DataWriter(s) of the data representation(s) supported by this Reader. This must be consistent with the TypeSupport associated with the Reader's **Topic** (p. 69).*

- **TypeConsistencyEnforcementQosPolicy type_consistency**

  Influences the algorithm that matches DataReaders and DataWriters based on their data type compatibility.
- **RpcQosPolicy rpc**

  Configure RPC relevant settings: instance_name, related_entity, and topic_aliases.
- **EntityNameQosPolicy entity_name**

  *Controls the **Entity** (p. 55) name of the **DataReader** (p. 26).*
- **LoggingQosPolicy logging**

  *Controls the logging behavior of the **DataReader** (p. 26).*
- **RTPSReaderQosPolicy rtps_reader**

  *Controls aspects of the RTPS Reader protocol for the **DataReader** (p. 26).*

3.1.1 Detailed Description

Structure that holds **DataReader** (p. 26) Quality of Service policies.

See also

DataReader::set_qos(DataReaderQos) set_qos()
DataReader::get_qos(DataReaderQos) get_qos()
Subscriber::create_datareader(TopicDescription topic, DataReaderQos qos, DataReaderListener listener, long mask) create_datareader()
Subscriber::set_default_datareader_qos(DataReaderQos) set_default_datareader_qos()
Subscriber::get_default_datareader_qos(DataReaderQos) get_default_datareader_qos()

3.2 DDS::DataWriterQos Class Reference

Structure that holds **DataWriter** (p. 37) Quality of Service policies.

Public Attributes

- **DurabilityQosPolicy durability**

  *The durability policy offered by the **DataWriter** (p. 37).*
- **DurabilityServiceQosPolicy durability_service**

  *The durability service configuration offered by the **DataWriter** (p. 37).*
- **DeadlineQosPolicy deadline**

  *The deadline committed to by the **DataWriter** (p. 37).*
- **LatencyBudgetQosPolicy latency_budget**

  *The latency allowed by the **DataWriter** (p. 37).*
- **LivelinessQosPolicy liveliness**

  *The liveliness mechanism offered by the **DataWriter** (p. 37).*
- **ReliabilityQosPolicy reliability**

  *The transport reliability offered by the **DataWriter** (p. 37).*
- **DestinationOrderQosPolicy destination_order**

  *The destination order logic offered by the **DataWriter** (p. 37).*
- **HistoryQosPolicy history**

  *The data history maintained by the **DataWriter** (p. 37).*
- **ResourceLimitsQosPolicy resource_limits**

*The resource limits set on the **DataWriter** (p. 37).*

- **TransportPriorityQosPolicy transport_priority**

  *The transport priority supported by the **DataWriter** (p. 37).*

- **LifespanQosPolicy lifespan**

  *The expiration time for old samples managed by the **DataWriter** (p. 37).*

- **UserDataQosPolicy user_data**

  *A sequence of octets associated with the **DataWriter** (p. 37).*

- **OwnershipQosPolicy ownership**

  *The type of 'ownership' offered by the **DataWriter** (p. 37).*

- **OwnershipStrengthQosPolicy ownership_strength**

  *The measure of 'ownership strength' offered by the **DataWriter** (p. 37).*

- **WriterDataLifecycleQosPolicy writer_data_lifecycle**

  *Indicates if unregistered instances should be automatically disposed by the **DataWriter** (p. 37).*

- **DataRepresentationQosPolicy representation**

  *Informs DataReader(s) of the single data representation supported by this Writer. This must be consistent with the TypeSupport associated with the Writer's **Topic** (p. 69).*

- **RpcQosPolicy rpc**

  Configure RPC relevant settings: instance_name, related_entity, and topic_aliases.

- **EntityNameQosPolicy entity_name**

  *Controls the **Entity** (p. 55) name of the **DataWriter** (p. 37).*

- **LoggingQosPolicy logging**

  *Controls the logging behavior of the **DataWriter** (p. 37).*

- **RTPSWriterQosPolicy rtps_writer**

  *Controls aspects of the RTPS Writer protocol for the **DataWriter** (p. 37).*

3.2.1 Detailed Description

Structure that holds **DataWriter** (p. 37) Quality of Service policies.

See also

DataWriter::set_qos(DataWriterQos) set_qos()
DataWriter::get_qos(DataWriterQos) get_qos()
Publisher::create_datawriter(Topic topic, DataWriterQos qos, DataWriterListener listener, long mask) create_datawriter()
Publisher::set_default_datawriter_qos(DataWriterQos) set_default_datawriter_qos()
Publisher::get_default_datawriter_qos(DataWriterQos) get_default_datawriter_qos()

3.3 DDS::DomainParticipantFactoryQos Class Reference

Structure that holds **DomainParticipantFactory** (p. 54) Quality of Service policies.

Public Attributes

- **EntityFactoryQosPolicy entity_factory**

  *Controls the behavior of the **DomainParticipant** (p. 44) 'create' operations.*

3.3.1 Detailed Description

Structure that holds **DomainParticipantFactory** (p. 54) Quality of Service policies.

See also

DomainParticipantFactory::set_qos(DomainParticipantFactoryQos) set_qos()
DomainParticipantFactory::get_qos(DomainParticipantFactoryQos) get_qos()

3.4 DDS::DomainParticipantQos Class Reference

Structure that holds **DomainParticipant** (p. 44) Quality of Service policies.

Public Attributes

- **UserDataQosPolicy user_data**

  *A sequence of octets associated with a **DomainParticipant** (p. 44).*
- **EntityFactoryQosPolicy entity_factory**

  *Controls the behavior of the **DomainParticipant** (p. 44) **create** operations.*
- **EntityNameQosPolicy entity_name**

  *Controls the **Entity** (p. 55) name of the **DomainParticipant** (p. 44).*
- **LoggingQosPolicy logging**

  *Controls the logging behavior of the **DomainParticipant** (p. 44).*
- **PeerParticipantQosPolicy peer_participants**

  *Specifies a list of **DomainParticipant** (p. 44) peers to attempt communication with. If empty, default Discovery is used.*
- **DiscoveryQosPolicy discovery**

  Controls aspects of the Discovery and Builtin entities.
- **ThreadModelQosPolicy thread_model**

  *Controls the threading behavior of the **DomainParticipant** (p. 44).*
- **PropertyQosPolicy properties**

  Additional name:value pair properties (if propagate=true, included in discovery)

3.4.1 Detailed Description

Structure that holds **DomainParticipant** (p. 44) Quality of Service policies.

See also

DomainParticipant::set_qos(DomainParticipantQos) set_qos()
DomainParticipant::get_qos(DomainParticipantQos) get_qos()
DomainParticipantFactory::create_participant(long domain_id, DomainParticipantQos qos, DomainParticipantListener listener, long mask) create_participant()
DomainParticipantFactory::set_default_participant_qos(DomainParticipantQos) set_default_participant_qos()
DomainParticipantFactory::get_default_participant_qos(DomainParticipantQos) get_default_participant_qos()

3.5 DDS::PublisherQos Class Reference

Structure that holds **Publisher** (p. 57) Quality of Service policies.

Public Attributes

- **PresentationQosPolicy presentation**

  Controls the presentation of groups of changes.
- **PartitionQosPolicy partition**

  Establishes a logical data partition.
- **GroupDataQosPolicy group_data**

  *A sequence of octets associated with the **Publisher** (p. 57).*
- **EntityFactoryQosPolicy entity_factory**

  *Controls the behavior of the **Publisher.create_datawriter()** (p. 59) operation.*
- **EntityNameQosPolicy entity_name**

  *Controls the **Entity** (p. 55) name of the **Publisher** (p. 57).*
- **LoggingQosPolicy logging**

  *Controls the logging behavior of the **Publisher** (p. 57).*

3.5.1 Detailed Description

Structure that holds **Publisher** (p. 57) Quality of Service policies.

See also

Publisher::set_qos(PublisherQos) set_qos()
Publisher::get_qos(PublisherQos) get_qos()
DomainParticipant::create_publisher(PublisherQos qos, PublisherListener listener, long mask) create_publisher()
DomainParticipant::set_default_publisher_qos(PublisherQos) set_default_publisher_qos()
DomainParticipant::get_default_publisher_qos(PublisherQos) get_default_publisher_qos()

3.5.2 Member Data Documentation

3.5.2.1 PartitionQosPolicy DDS::PublisherQos::partition

Establishes a logical data partition.

DataWriters and DataReaders that are 'in' the same partition (ie, the partition of the containing **Publisher** (p. 57) and **Subscriber** (p. 65) match) can communicate. If the partitions do not match, then they cannot communicate.

3.5.2.2 PresentationQosPolicy DDS::PublisherQos::presentation

Controls the presentation of groups of changes.

See also

Publisher::begin_coherent_changes() (p. 58) begin_coherent_changes()
Publisher::end_coherent_changes() (p. 60) end_coherent_changes()
Subscriber::begin_access() (p. 66) begin_access()
Subscriber::end_access() (p. 66) end_access()

3.6 DDS::SubscriberQos Class Reference

Structure that holds **Subscriber** (p. 65) Quality of Service policies.

Public Attributes

- **PresentationQosPolicy presentation**

  Controls the presentation of groups of changes.
- **PartitionQosPolicy partition**

  Establishes a logical data partition.
- **GroupDataQosPolicy group_data**

  *A sequence of octets associated with the **Publisher** (p. 57).*
- **EntityFactoryQosPolicy entity_factory**

  *Controls the behavior of the **Subscriber.create_datareader()** (p. 67) operation.*
- **EntityNameQosPolicy entity_name**

  *Controls the **Entity** (p. 55) name of the **Subscriber** (p. 65).*
- **LoggingQosPolicy logging**

  *Controls the logging behavior of the **Subscriber** (p. 65).*

3.6.1 Detailed Description

Structure that holds **Subscriber** (p. 65) Quality of Service policies.

See also

Subscriber::set_qos(SubscriberQos)
Subscriber::get_qos(SubscriberQos)
DomainParticipant::create_subscriber(SubscriberQos qos, SubscriberListener listener, long mask)
DomainParticipant::set_default_subscriber_qos(SubscriberQos)
DomainParticipant::get_default_subscriber_qos(SubscriberQos)

3.6.2 Member Data Documentation

3.6.2.1 PartitionQosPolicy DDS::SubscriberQos::partition

Establishes a logical data partition.

DataWriters and DataReaders that are 'in' the same partition (ie, the partition of the containing **Publisher** (p. 57) and **Subscriber** (p. 65) match) can communicate. If the partitions do not match, then they cannot communicate.

3.6.2.2 PresentationQosPolicy DDS::SubscriberQos::presentation

Controls the presentation of groups of changes.

See also

Publisher::begin_coherent_changes() (p. 58)
Publisher::end_coherent_changes() (p. 60)
Subscriber::begin_access() (p. 66)
Subscriber::end_access() (p. 66)

3.7 DDS::TopicQos Class Reference

Structure that holds **Topic** (p. 69) Quality of Service policies.

Public Attributes

- **TopicDataQosPolicy topic_data**

  *A sequence of octets associated with a **Topic** (p. 69).*
- **DurabilityQosPolicy durability**

  *The durability policy of the **Topic** (p. 69).*
- **DurabilityServiceQosPolicy durability_service**

  Configures the service supporting the TRANSIENT and PERSITENT durability kinds.
- **DeadlineQosPolicy deadline**

  *Defines the expected update frequency for data instances within the **Topic** (p. 69).*
- **LatencyBudgetQosPolicy latency_budget**

  *Identifies the 'urgency' of the data on the **Topic** (p. 69). The middleware uses this to batch data samples is possible.*
- **LivelinessQosPolicy liveliness**

  *Identifies the mechanism used to detect and maintain livelines of DataWriters on the **Topic** (p. 69).*
- **ReliabilityQosPolicy reliability**

  *Indicates the level of transport reliability on the **Topic** (p. 69).*
- **DestinationOrderQosPolicy destination_order**

  *The ordering of received samples on the **Topic** (p. 69) will be either by SOURCE or RECEPTION timestamp.*
- **HistoryQosPolicy history**

  *The amount of historical data maintained for the **Topic** (p. 69).*
- **ResourceLimitsQosPolicy resource_limits**

  *The resource limitations for the **Topic** (p. 69).*
- **TransportPriorityQosPolicy transport_priority**

  *The priority to be used for messages on the **Topic** (p. 69).*
- **LifespanQosPolicy lifespan**

  *The 'expiration time' for old data samples on the **Topic** (p. 69).*
- **OwnershipQosPolicy ownership**

  *The type of 'ownership' expected for data instances on the **Topic** (p. 69).*
- **DataRepresentationQosPolicy representation**

  The data representation(s) supported by Writer(s) and Reader(s) of this topic. The first entry in the 'value' sequence corresponds to the Writer(s), all entries apply to Reader(s).
- **EntityNameQosPolicy entity_name**

  *Controls the **Entity** (p. 55) name of the **Topic** (p. 69).*
- **LoggingQosPolicy logging**

  *Controls the logging behavior of the **Topic** (p. 69).*

3.7.1 Detailed Description

Structure that holds **Topic** (p. 69) Quality of Service policies.

See also

Topic::set_qos(TopicQos) set_qos()
Topic::get_qos(TopicQos) get_qos()
DomainParticipant::create_topic (p. 48)(java.lang.String topic_name, java.lang.String type_name, **TopicQos** (p. 80) **DDS Quality of Service** (p. 4), **TopicListener** (p. 112) listener, long mask) create_topic()
DomainParticipant::set_default_topic_qos(TopicQos) set_default_topic_qos()

Chapter 4

QoS Policies

4.1 DDS::DataRepresentationQosPolicy Struct Reference

Describes the data representation used by a topic.

Public Member Functions

- **DataRepresentationQosPolicy** ()

Public Attributes

- DDS::DataRepresentationIdSeq **value**

  data representation

4.1.1 Detailed Description

Describes the data representation used by a topic.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 DDS::DataRepresentationQosPolicy::DataRepresentationQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.2 DDS::DeadlineQosPolicy Struct Reference

This QoS policy establishes a minimum update period for data instances.

Public Member Functions

- **DeadlineQosPolicy** ()

Public Attributes

- struct **DDS::Duration_t period**

  the maximum interval between instance updates.

4.2.1 Detailed 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** (p. 26) 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** (p. 37) fails to meet the update period, then the **DataWriter** (p. 37) is notified by a call to the offered_deadline_missed listener, and the **DataReader** (p. 26) is notified by a call to the requested_deadline_missed listener.

See also

DataReaderListener::on_requested_deadline_missed(DataReader the_reader, RequestedDeadlineMissedStatus status) on_requested_deadline_missed
DataWriterListener::on_offered_deadline_missed(DataWriter writer, OfferedDeadlineMissedStatus status) on_offered_deadline_missed

4.2.2 Constructor & Destructor Documentation

4.2.2.1 DDS::DeadlineQosPolicy::DeadlineQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.3 DDS::DestinationOrderQosPolicy Struct Reference

This QoS policy controls how each **Subscriber** (p. 65) orders received data samples.

Public Member Functions

- **DestinationOrderQosPolicy** ()

Public Attributes

- enum **DDS::DestinationOrderQosPolicyKind kind**

  the kind of ordering applied to samples

4.3.1 Detailed Description

This QoS policy controls how each **Subscriber** (p. 65) 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** (p. 37)).

CoreDX DDS currently implements only BY_RECEPTION_TIMESTAMP.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 DDS::DestinationOrderQosPolicy::DestinationOrderQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.4 DiscoveryQosPolicy Struct Reference

QoS Policy for configuring aspects of the Discovery and Builtin entities.

Public Attributes

- **DDS::DiscoveryQosPolicyDiscoveryKind discovery_kind**

  the kind of discovery to employ
- DDS_BUILTIN_TOPIC_KEY_TYPE_NATIVE **guid_pid**

  specify the PID field of the GUID, leave as zero (0) for automatic
- Duration_t **dpd_push_period**

  Multicast DiscoveredParticipantData each period.
- Duration_t **dpd_lease_duration**

  How long we consider a discoverd Participant to be alive without hearing from them.
- Duration_t **heartbeat_period**

  The heartbeat_period to configure for builtin writers.
- Duration_t **nack_response_delay**

  The nack_response_delay to configure for builtin writers.
- Duration_t **nack_suppress_delay**

  The nack_suppress_delay to configure for builtin writers.
- unsigned int **min_buffer_size**

  The min_buffer_size to configure for builtin writers.
- unsigned int **max_buffer_size**

  The max_buffer_size to configure for builtin writers.
- Duration_t **heartbeat_response_delay**

  The heartbeat_response_delay to configure for builtin readers.
- unsigned char **send_initial_nack**

  send a ACKNACK immediately after matching with a new Writer
- unsigned char **send_msglen_submsg**

  include 'msglen' submessage in RTPS header

4.4.1 Detailed Description

QoS Policy for configuring aspects of the Discovery and Builtin entities.

4.5 DDS::DurabilityQosPolicy Struct Reference

The **DurabilityQosPolicy** (p. 85) controls the durablity of data.

Public Member Functions

- **DurabilityQosPolicy** ()

Public Attributes

- enum **DDS::DurabilityQosPolicyKind kind**

  the durability kind

4.5.1 Detailed Description

The **DurabilityQosPolicy** (p. 85) 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** (p. 37). If that **DataWriter** (p. 37) 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** (p. 37), and is available even after that **DataWriter** (p. 37) 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** (p. 37) can offer to provide a level of durablity for data that it generates, and a **DataReader** (p. 26) requests the level of durability that it requires. If the **DataReader** (p. 26) requests a 'higher' level of durability than that offered by the **DataWriter** (p. 37), then the QoS policy is incompatible.

CoreDX DDS currently supports VOLATILE and TRANSIENT_LOCAL.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 DDS::DurabilityQosPolicy::DurabilityQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.6 DDS::DurabilityServiceQosPolicy Struct Reference

Public Member Functions

- **DurabilityServiceQosPolicy** ()

Public Attributes

- struct **DDS::Duration_t service_cleanup_delay**

  service cleanup delay
- enum **DDS::HistoryQosPolicyKind history_kind**

  history kind
- int **history_depth**

history depth

- int **max_samples**

  maximum samples limit for the service

- int **max_instances**

  maximum instances limit for the service

- int **max_samples_per_instance**

  maximum samples per instance limit for the service

4.6.1 Detailed Description

The **DurabilityServiceQosPolicy** (p. 86) 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** (p. 37). If that **DataWriter** (p. 37) 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** (p. 37), and is available even after that **DataWriter** (p. 37) 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** (p. 37) 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.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 DDS::DurabilityServiceQosPolicy::DurabilityServiceQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.7 EntityFactoryQosPolicy Struct Reference

Public Attributes

- unsigned char **autoenable_created_entities**

  should created entities be automatically enabled

4.7.1 Detailed 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.

4.8 EntityNameQosPolicy Struct Reference

Public Attributes

- char **value** [COREDX_ENTITY_NAME_MAX]

  string to use as entity name

4.8.1 Detailed Description

Assigns a name to an Entity. This is a CoreDX DDS extension, and is provided for convenience. Entity names assigned through this QoS are not used internally by the middleware, they are provided for debug and analysis purposes.

4.9 DDS::GroupDataQosPolicy Struct Reference

Allows the application to attach arbitrary information to a **Publisher** (p. 57) or **Subscriber** (p. 65).

Public Member Functions

- **GroupDataQosPolicy** ()

Public Attributes

- **DDS::OctetSeq value**

  sequence of bytes

4.9.1 Detailed Description

Allows the application to attach arbitrary information to a **Publisher** (p. 57) or **Subscriber** (p. 65).

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** (p. 107) and **DataWriterListener** (p. 109) to implement conditional matching policies.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 DDS::GroupDataQosPolicy::GroupDataQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.10 DDS::HistoryQosPolicy Struct Reference

Controls the ammount of historical data maintained by a **DataReader** (p. 26) or **DataWriter** (p. 37).

Public Member Functions

- **HistoryQosPolicy** ()

Public Attributes

- enum **DDS::HistoryQosPolicyKind kind**

  history kind
- int **depth**

  history depth

4.10.1 Detailed Description

Controls the ammount of historical data maintained by a **DataReader** (p. 26) or **DataWriter** (p. 37).

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

ResourceLimitsQosPolicy (p. 97)

4.10.2 Constructor & Destructor Documentation

4.10.2.1 DDS::HistoryQosPolicy::HistoryQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.11 DDS::LatencyBudgetQosPolicy Struct Reference

Specifies allowable latency.

Public Member Functions

- **LatencyBudgetQosPolicy** ()

Public Attributes

- struct **DDS::Duration_t duration**

  the maximum delay before processing.

4.11.1 Detailed 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** (p. 37) must offer a latency_budget that is equal to or smaller than that requested by the **DataReader** (p. 26).

4.11.2 Constructor & Destructor Documentation

4.11.2.1 DDS::LatencyBudgetQosPolicy::LatencyBudgetQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.12 DDS::LifespanQosPolicy Struct Reference

Specifies the maximum duration of validity of the data written by the **DataWriter** (p. 37).

Public Member Functions

- **LifespanQosPolicy** ()

Public Attributes

- struct **DDS::Duration_t duration**

  lifespan duration

4.12.1 Detailed Description

Specifies the maximum duration of validity of the data written by the **DataWriter** (p. 37).

The default value of the lifespan duration is infinite.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 DDS::LifespanQosPolicy::LifespanQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.13 DDS::LivelinessQosPolicy Struct Reference

Determines the mechanism and parameters used by the application to determine whether an **Entity** (p. 55) is alive.

Public Member Functions

- **LivelinessQosPolicy** ()

Public Attributes

- enum **DDS::LivelinessQosPolicyKind kind**

  the kind of liveliness mechanism
- struct **DDS::Duration_t lease_duration**

  the duration of the liveliness lease

4.13.1 Detailed Description

Determines the mechanism and parameters used by the application to determine whether an **Entity** (p. 55) 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** (p. 109) and **DataReaderListener.on_liveliness_changed** (p. 108) listeners.

See also

DataReaderListener::on_liveliness_changed (p. 108)(**DataReader** (p. 26) the_reader, LivelinessChangedStatus **DDS Status Structures** (p. 13)) on_liveliness_changed
DataWriterListener::on_liveliness_lost (p. 109)(**DataWriter** (p. 37) writer, LivelinessLostStatus **DDS Status Structures** (p. 13)) on_liveliness_lost

4.13.2 Constructor & Destructor Documentation

4.13.2.1 DDS::LivelinessQosPolicy::LivelinessQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.14 LoggingQosPolicy Struct Reference

Controls the amount and kind of information that is logged.

Public Attributes

- unsigned int **flags**

  flags to control logging output

4.14.1 Detailed Description

Controls the amount and kind of information that is logged.

The application must be linked with the instrumented library in order to enable loggin with this **LoggingQosPolicy** (p. 91).

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.

Refer to include/dds/coredx_logging.h for the numeric values for the flags

4.15 DDS::OwnershipQosPolicy Struct Reference

Determines instance ownership in the case of multple writers. CoreDX DDS supports both SHARED_OWNERSHIP_QOS and EXCLUSIVE_OWNERSHIP_QOS.

Public Member Functions

- **OwnershipQosPolicy** ()

Public Attributes

- enum **DDS::OwnershipQosPolicyKind kind**

  the ownership kind

4.15.1 Detailed Description

Determines instance ownership in the case of multple writers. CoreDX DDS supports both SHARED_OWNERSHIP_QOS and EXCLUSIVE_OWNERSHIP_QOS.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 DDS::OwnershipQosPolicy::OwnershipQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.16 DDS::OwnershipStrengthQosPolicy Struct Reference

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 Member Functions

- **OwnershipStrengthQosPolicy** ()

Public Attributes

- int **value**

  the ownership strength value

4.16.1 Detailed 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.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 DDS::OwnershipStrengthQosPolicy::OwnershipStrengthQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.17 DDS::PartitionQosPolicy Struct Reference

Defines a logical data partition.

Public Member Functions

- **PartitionQosPolicy** ()

Public Attributes

- **DDS::StringSeq name**

  sequence of partition names

4.17.1 Detailed Description

Defines a logical data partition.

This QoS is assigned to a **Publisher** (p. 57) or **Subscriber** (p. 65). 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 "".

4.17.2 Constructor & Destructor Documentation

4.17.2.1 DDS::PartitionQosPolicy::PartitionQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.18 PeerParticipantQosPolicy Struct Reference

Configures a list of DomainParticipant peers to attempt communication with.

Public Attributes

- ParticipantLocatorSeq **value**

  sequence of peer locators with which we should attempt communication
- unsigned char **strict_match**

4.18.1 Detailed Description

Configures a list of DomainParticipant peers to attempt communication with.

If the value list is empty, then default Discovery is used.

4.18.2 Member Data Documentation

4.18.2.1 unsigned char PeerParticipantQosPolicy::strict_match

restrict matching to those peers listed (default is 0/FALSE)

4.19 DDS::PresentationQosPolicy Struct Reference

Controls the presentation of received data samples to the application. CoreDX DDS currently supports only the access_scope = INSTANCE_PRESENTATION_QOS policy.

Public Member Functions

- **PresentationQosPolicy** ()

Public Attributes

- enum **DDS::PresentationQosPolicyAccessScopeKind access_scope**

  the 'scope' of the presentation policy. Determines the extent to which the sample 'coherency' and 'order' are maintained
- unsigned char **coherent_access**

  Determines if coherent access is supported within the defined 'scope'.
- unsigned char **ordered_access**

  Determines if ordered access is supported within the defined 'scope'.

4.19.1 Detailed Description

Controls the presentation of received data samples to the application. CoreDX DDS currently supports only the access_scope = INSTANCE_PRESENTATION_QOS policy.

4.19.2 Constructor & Destructor Documentation

4.19.2.1 DDS::PresentationQosPolicy::PresentationQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.20 DDS::PropertyQosPolicy Struct Reference

Public Member Functions

- **PropertyQosPolicy** ()

Public Attributes

- DDS::PropertySeq **value**

  *sequence of **Property_t** (p. 132)'s*

4.20.1 Detailed Description

The **PropertyQosPolicy** (p. 95) 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.

4.20.2 Constructor & Destructor Documentation

4.20.2.1 DDS::PropertyQosPolicy::PropertyQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.21 ReaderDataLifecycleQosPolicy Struct Reference

Specifies the lifecycle behavior of data instances managed by the DataReader.

Public Attributes

- Duration_t **autopurge_nowriter_samples_delay**

  delay after which it is ok to purge samples from instances with no writer(s)
- Duration_t **autopurge_disposed_samples_delay**

  delay after which it is ok to purge samples from instances that are disposed

4.21.1 Detailed 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.

4.22 DDS::ReliabilityQosPolicy Struct Reference

Indicates the level of reliability offered/provided by the **Entity** (p. 55). 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.

Public Member Functions

- **ReliabilityQosPolicy** ()

Public Attributes

- enum **DDS::ReliabilityQosPolicyKind kind**

  reliability kind
- struct **DDS::Duration_t max_blocking_time**

  maximum blocking time

4.22.1 Detailed Description

Indicates the level of reliability offered/provided by the **Entity** (p. 55). 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** (p. 88), **ResourceLimitsQosPolicy** (p. 97), and the **DurabilityQosPolicy** (p. 85).

Note: If samples are removed from the history cache, then they are not be available to be resent.

See also

HistoryQosPolicy (p. 88)
ResourceLimitsQosPolicy (p. 97)
DurabilityQosPolicy (p. 85)

4.22.2 Constructor & Destructor Documentation

4.22.2.1 DDS::ReliabilityQosPolicy::ReliabilityQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.23 DDS::ResourceLimitsQosPolicy Struct Reference

Specifies the resources that the Service can use to maintain data samples and instances.

Public Member Functions

- **ResourceLimitsQosPolicy** ()

Public Attributes

- int **max_samples**

  maximum samples allowed in the cache
- int **max_instances**

  maximum instances allowed in the cache
- int **max_samples_per_instance**

  maximum samples per instance allowed in the cache
- unsigned char **preallocate_samples**

  *Request that the Reader or Writer pre-allocate the specified number of samples (if max_samples is not unlimited). CoreDX **DDS** (p. 313) Extension.*
- unsigned char **preallocate_instances**

  *Request that the Reader or Writer pre-allocate the specified number of instances (if max_instances is not unlimited). CoreDX **DDS** (p. 313) Extension.*

4.23.1 Detailed Description

Specifies the resources that the Service can use to maintain data samples and instances.

See also

ReliabilityQosPolicy (p. 96)
HistoryQosPolicy (p. 88)

4.23.2 Constructor & Destructor Documentation

4.23.2.1 DDS::ResourceLimitsQosPolicy::ResourceLimitsQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.24 DDS::RpcQosPolicy Struct Reference

Augment a **DataWriter** (p. 37) or **DataReader** (p. 26) with RPC specific information.

Public Member Functions

- **RpcQosPolicy** ()

Public Attributes

- fixedstring255_t **service_instance_name**

  the name of the service instance
- struct **DDS::GUID_t related_entity_guid**

  guid of the related entity (request / reply)
- **DDS::StringSeq topic_aliases**

  a sequence of aliases to support derived interfaces

4.24.1 Detailed Description

Augment a **DataWriter** (p. 37) or **DataReader** (p. 26) with RPC specific information.

4.24.2 Constructor & Destructor Documentation

4.24.2.1 DDS::RpcQosPolicy::RpcQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.25 RTPSReaderQosPolicy Struct Reference

QoS Policy for configuring aspects of the RTPS Reader Protocol.

Public Attributes

- Duration_t **heartbeat_response_delay**

  ammount of time allowed for responding to a heartbeat
- unsigned char **accept_batch_msg**

  allow writers to use the 'BATCH' RTPS message
- unsigned char **send_typecode**

  send TYPECODE info for associated data type
- unsigned char **send_initial_nack**

  send a ACKNACK immediately after matching with a new Writer
- unsigned int **precache_max_samples**

  Maximum number of samples held in pre-cache [only for RELIABLE readers].

4.25.1 Detailed Description

QoS Policy for configuring aspects of the RTPS Reader Protocol.

4.26 RTPSWriterQosPolicy Struct Reference

QoS Policy for configuring aspects of the RTPS Writer Protocol.

Public Attributes

- Duration_t **heartbeat_period**

  period to transmit heartbeat messages if required
- Duration_t **nack_response_delay**

  ammount of time allowed for responding to NACKs
- Duration_t **nack_suppress_delay**

  ammount of time to ignore NACKs after a repair
- Duration_t **ack_deadline**

  after which a realiable reader will be considered unresponsive
- unsigned int **min_buffer_size**

  min size in bytes of written data
- unsigned int **max_buffer_size**

  max size in bytes of written data
- unsigned char **apply_filters**

  apply ContentFilter(s) at the writer (writer side filtering)
- unsigned char **enable_batch_msg**

  use the 'BATCH' RTPS message to send data if all Readers accept BATCH
- unsigned char **send_typecode**

  send TYPECODE info for associated data type

4.26.1 Detailed Description

QoS Policy for configuring aspects of the RTPS Writer Protocol.

4.27 ThreadModelQosPolicy Struct Reference

QoS Policy for configuring the threading behavior of the DomainParticipant.

Public Attributes

- unsigned char **use_threads**

  should the DomainParticipant use threads for internal work
- unsigned char **create_listener_thread**

  should the DomainParticipant create a separate thread for invoking listener callback routines

4.27.1 Detailed Description

QoS Policy for configuring the threading behavior of the DomainParticipant.

4.28 DDS::TimeBasedFilterQosPolicy Struct Reference

Defines a filter based on time between samples. The **DataReader** (p. 26) indicates that it wants at most one sample for each instance every minimum_separation interval.

Public Member Functions

- **TimeBasedFilterQosPolicy** ()

Public Attributes

- struct **DDS::Duration_t minimum_separation**

  the minimum separation between instance updates

4.28.1 Detailed Description

Defines a filter based on time between samples. The **DataReader** (p. 26) indicates that it wants at most one sample for each instance every minimum_separation interval.

It is inconsistent for a **DataReader** (p. 26) to have a minimum_separation larger than its DEADLINE period. By default minimum_separation=0 indicating that the **DataReader** (p. 26) should be sent all values.

4.28.2 Constructor & Destructor Documentation

4.28.2.1 DDS::TimeBasedFilterQosPolicy::TimeBasedFilterQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.29 DDS::TopicDataQosPolicy Struct Reference

Allows the application to attach arbitrary information to a **Topic** (p. 69) QoS.

Public Member Functions

- **TopicDataQosPolicy** ()

Public Attributes

- **DDS::OctetSeq value**

  sequence of bytes

4.29.1 Detailed Description

Allows the application to attach arbitrary information to a **Topic** (p. 69) 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** (p. 112) to implement conditional matching policies.

4.29.2 Constructor & Destructor Documentation

4.29.2.1 DDS::TopicDataQosPolicy::TopicDataQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.30 DDS::TransportPriorityQosPolicy Struct Reference

A hint to the middleware to help configure the transport priority mechanism.

Public Member Functions

- **TransportPriorityQosPolicy** ()

Public Attributes

- int **value**

  priority value

4.30.1 Detailed Description

A hint to the middleware to help configure the transport priority mechanism.

4.30.2 Constructor & Destructor Documentation

4.30.2.1 DDS::TransportPriorityQosPolicy::TransportPriorityQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.31 DDS::TypecodeQosPolicy Struct Reference

Typecode representing the datatype a **DataReader** (p. 26) reads or a **DataWriter** (p. 37) writes.

Public Member Functions

- **TypecodeQosPolicy** ()

Public Attributes

- **DDS::OctetSeq value**

  sequence of bytes representing the TypeCode for the data type
- unsigned char **encoding**

  encoding of the typecode bytes (little endian / big endian

4.31.1 Detailed Description

Typecode representing the datatype a **DataReader** (p. 26) reads or a **DataWriter** (p. 37) writes.

4.31.2 Constructor & Destructor Documentation

4.31.2.1 DDS::TypecodeQosPolicy::TypecodeQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.32 DDS::TypeConsistencyEnforcementQosPolicy Struct Reference

rules for determining type consistency

Public Member Functions

- **TypeConsistencyEnforcementQosPolicy** ()

Public Attributes

- enum **DDS::TypeConsistencyKind kind**

  type consistency kind

4.32.1 Detailed Description

rules for determining type consistency

The Type Consistency Enforcement QoS Policy defines the rules for determining whether the type used to publish a given data stream is consistent with that used to subscribe to it. It applies to DataReaders.

4.32.2 Constructor & Destructor Documentation

4.32.2.1 DDS::TypeConsistencyEnforcementQosPolicy::TypeConsistencyEnforcementQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.33 DDS::UserDataQosPolicy Struct Reference

Allows the application to attach arbitrary information to a **DomainParticipant** (p. 44), **DataWriter** (p. 37) or **DataReader** (p. 26) QoS.

Public Member Functions

- **UserDataQosPolicy** ()

Public Attributes

- **DDS::OctetSeq value**

  sequence of bytes

4.33.1 Detailed Description

Allows the application to attach arbitrary information to a **DomainParticipant** (p. 44), **DataWriter** (p. 37) or **DataReader** (p. 26) QoS.

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.

4.33.2 Constructor & Destructor Documentation

4.33.2.1 DDS::UserDataQosPolicy::UserDataQosPolicy ( )

Constructor, Copy Constructor, Destructor, Assignment operator

4.34 WriterDataLifecycleQosPolicy Struct Reference

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

- unsigned char **autodispose_unregistered_instances**

  should the writer dispose instances that are unregistered

4.34.1 Detailed 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.

Chapter 5

QoS Provider

5.1 DDS::QosProvider Class Reference

QosProvider (p. 105) loads QoS settings from a library and provides interfaces to access entity specific QoS Policies.

Public Member Functions

- **QosProvider** (const char ∗uri, const char ∗profile, **CDX_XmlApi** ∗xml_parser)

  *Constructor. Creates a new **QosProvider** (p. 105) instance that loads QoS settings from a library.*
- ∼**QosProvider** ()

  *Destructor. Returns a **QosProvider** (p. 105) back to the factory.*
- **DomainParticipantFactoryQos** ∗ **get_participantfactory_qos** (const char ∗id)

  *Access a **DomainParticipantFactory** (p. 54) QoS.*
- void **return_participantfactory_qos** (**DomainParticipantFactoryQos** ∗qos)

  Return a DomainParticipanFactoryQos back to the factory.
- **DomainParticipantQos** ∗ **get_participant_qos** (const char ∗id)

  *Access a **DomainParticipant** (p. 44) QoS.*
- void **return_participant_qos** (**DomainParticipantQos** ∗qos)

  *Return a **DomainParticipantQos** (p. 78) back to the factory.*
- **TopicQos** ∗ **get_topic_qos** (const char ∗id)

  *Access a **Topic** (p. 69) QoS.*
- void **return_topic_qos** (**TopicQos** ∗qos)

  *Return a **TopicQos** (p. 80) back to the factory.*
- **SubscriberQos** ∗ **get_subscriber_qos** (const char ∗id)

  *Access a **Subscriber** (p. 65) QoS.*
- void **return_subscriber_qos** (**SubscriberQos** ∗qos)

  *Return a **SubscriberQos** (p. 79) back to the factory.*
- **PublisherQos** ∗ **get_publisher_qos** (const char ∗id)

  *Access a **Publisher** (p. 57) QoS.*
- void **return_publisher_qos** (**PublisherQos** ∗qos)

  *Return a **PublisherQos** (p. 78) back to the factory.*
- **DataReaderQos** ∗ **get_datareader_qos** (const char ∗id)

  *Access a **DataReader** (p. 26) QoS.*

- void **return_datareader_qos** (**DataReaderQos** ∗qos)

  *Return a **DataReaderQos** (p. 75) back to the factory.*
- **DataWriterQos** ∗ **get_datawriter_qos** (const char ∗id)

  *Access a **DataWriter** (p. 37) QoS.*
- void **return_datawriter_qos** (**DataWriterQos** ∗qos)

  *Return a **DataWriterQos** (p. 76) back to the factory.*

5.1.1 Detailed Description

QosProvider (p. 105) loads QoS settings from a library and provides interfaces to access entity specific QoS Policies.

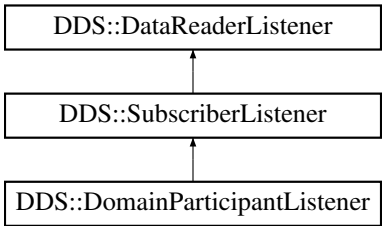
Chapter 6

DDS Listeners

6.1 DDS::DataReaderListener Class Reference

The **DataReaderListener** (p. 107) provides asynchronous notification of **DataReader** (p. 26) events.

Inheritance diagram for DDS::DataReaderListener:



Public Member Functions

- virtual void **on_requested_deadline_missed** (**DataReader** ∗, const RequestedDeadlineMissedStatus &)
- virtual void **on_requested_incompatible_qos** (**DataReader** ∗, const RequestedIncompatibleQosStatus &)
- virtual void **on_sample_rejected** (**DataReader** ∗, const SampleRejectedStatus &)
- virtual void **on_liveliness_changed** (**DataReader** ∗, const LivelinessChangedStatus &)
- virtual void **on_data_available** (**DataReader** ∗)
- virtual void **on_subscription_matched** (**DataReader** ∗, const SubscriptionMatchedStatus &)
- virtual void **on_sample_lost** (**DataReader** ∗, const SampleLostStatus &)

6.1.1 Detailed Description

The **DataReaderListener** (p. 107) provides asynchronous notification of **DataReader** (p. 26) events.

This listener can be installed during **DataReader** (p. 26) 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** (p. 44).

6.1.2 Member Function Documentation

6.1.2.1 virtual void DDS::DataReaderListener::on_data_available (DataReader *) [inline],[virtual]

on_data_available() (p. 108) is called when the CoreDX DDS middleware detects that new data or data state information is available.

The **status** argument provides a snapshot of the status at the time the listener was invoked.

6.1.2.2 virtual void DDS::DataReaderListener::on_liveliness_changed (DataReader *, const LivelinessChangedStatus &) [inline],[virtual]

on_liveliness_changed() (p. 108) is called when the CoreDX DDS middleware detects that the liveliness of a matched **DataWriter** (p. 37) has changed (either 'active' or 'inactive').

The **status** argument provides a snapshot of the status at the time the listener was invoked.

6.1.2.3 virtual void DDS::DataReaderListener::on_requested_deadline_missed (DataReader *, const RequestedDeadlineMissedStatus &) [inline],[virtual]

on_requested_deadline_missed() (p. 108) is called when the CoreDX DDS middleware detects that the deadline specified in the **DataReader** (p. 26) QoS DEADLINE policy was not satisfied for a data instance.

The **status** argument provides a snapshot of the status at the time the listener was invoked.

6.1.2.4 virtual void DDS::DataReaderListener::on_requested_incompatible_qos (DataReader *, const RequestedIncompatibleQosStatus &) [inline],[virtual]

on_requested_incompatible_qos() (p. 108) is called when the CoreDX DDS middleware detects that the **DataReader** (p. 26) requested a QoS policy that was incompatible with that offered by a **DataWriter** (p. 37).

The **status** argument provides a snapshot of the status at the time the listener was invoked.

6.1.2.5 virtual void DDS::DataReaderListener::on_sample_lost (DataReader *, const SampleLostStatus &) [inline], [virtual]

on_sample_lost() (p. 108) is called when the CoreDX DDS middleware detects that a sample has been lost (never received).

The **status** argument provides a snapshot of the status at the time the listener was invoked.

6.1.2.6 virtual void DDS::DataReaderListener::on_sample_rejected (DataReader *, const SampleRejectedStatus &) [inline],[virtual]

on_sample_rejected() (p. 108) is called when the CoreDX DDS middleware detects that a sample has been rejected.

The **status** argument provides a snapshot of the status at the time the listener was invoked.

6.1.2.7 virtual void DDS::DataReaderListener::on_subscription_matched (DataReader *, const SubscriptionMatchedStatus &) [inline],[virtual]

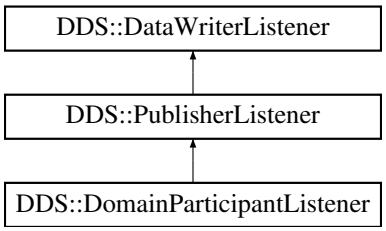
on_subscription_matched() (p. 108) is called when the CoreDX DDS middleware detects that the **DataReader** (p. 26) has matched or ceased to be matched with a **DataWriter** (p. 37).

The **status** argument provides a snapshot of the status at the time the listener was invoked.

6.2 DDS::DataWriterListener Class Reference

The **DataWriterListener** (p. 109) provides asynchronous notification of **DataWriter** (p. 37) events.

Inheritance diagram for DDS::DataWriterListener:



Public Member Functions

- virtual void **on_offered_deadline_missed** (**DataWriter** ∗, const OfferedDeadlineMissedStatus &)
- virtual void **on_offered_incompatible_qos** (**DataWriter** ∗, const OfferedIncompatibleQosStatus &)
- virtual void **on_liveliness_lost** (**DataWriter** ∗, const LivelinessLostStatus &)
- virtual void **on_publication_matched** (**DataWriter** ∗, const PublicationMatchedStatus &)

6.2.1 Detailed Description

The **DataWriterListener** (p. 109) provides asynchronous notification of **DataWriter** (p. 37) events.

This listener can be installed during **DataWriter** (p. 37) 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** (p. 44).

6.2.2 Member Function Documentation

6.2.2.1 virtual void DDS::DataWriterListener::on_liveliness_lost (DataWriter ∗ , const LivelinessLostStatus &) [inline], [virtual]

on_liveliness_lost() (p. 109) is called when the CoreDX DDS middleware detects that the **DataWriter** (p. 37) has not satisfied its LIVELINESS QoS setting.

The **status** argument provides a snapshot of the status at the time the listener was invoked.

6.2.2.2 virtual void DDS::DataWriterListener::on_offered_deadline_missed (DataWriter ∗ , const OfferedDeadlineMissedStatus &) [inline], [virtual]

on_offered_deadline_missed() (p. 109) is called when the CoreDX DDS middleware detects that the deadline that the **DataWriter** (p. 37) has offered Through the DEADLINE QoS was not satisfied. That is, the **DataWriter** (p. 37) has failed to update an instance with the frequency specified in the DEADLINE QoS.

The **status** argument provides a snapshot of the status at the time the listener was invoked.

6.2.2.3 virtual void DDS::DataWriterListener::on_offered_incompatible_qos (DataWriter *, const OfferedIncompatibleQosStatus &) [inline],[virtual]

on_offered_incompatible_qos() (p. 110) is called when the CoreDX DDS middleware detects that the **DataWriter** (p. 37) has offered a QoS policy setting that is incompatible with that requested by a potentially matching **DataReader** (p. 26).

The **status** argument provides a snapshot of the status at the time the listener was invoked.

6.2.2.4 virtual void DDS::DataWriterListener::on_publication_matched (DataWriter *, const PublicationMatchedStatus &) [inline],[virtual]

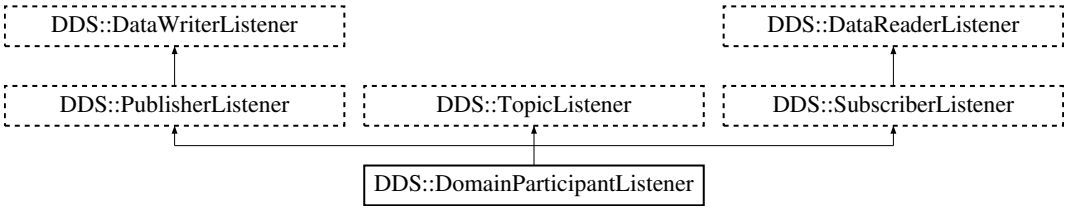
on_publication_matched() (p. 110) is called when the CoreDX DDS middleware detects that the **DataWriter** (p. 37) has matched with a **DataReader** (p. 26) or has ceased to be matched with a **DataReader** (p. 26).

The **status** argument provides a snapshot of the status at the time the listener was invoked.

6.3 DDS::DomainParticipantListener Class Reference

The **DomainParticipantListener** (p. 110) provides asynchronous notification of **DomainParticipant** (p. 44) events.

Inheritance diagram for DDS::DomainParticipantListener:



Additional Inherited Members

6.3.1 Detailed Description

The **DomainParticipantListener** (p. 110) provides asynchronous notification of **DomainParticipant** (p. 44) events.

This listener can be installed during **DomainParticipant** (p. 44) creation, DomainParticipantFactory_create_participant(), as well as by calling DomainParticipant_set_listener().

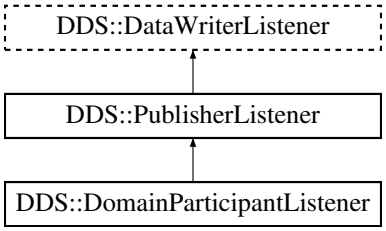
Note

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** (p. 44).

6.4 DDS::PublisherListener Class Reference

The **PublisherListener** (p. 110) provides asynchronous notification of **Publisher** (p. 57) events.

Inheritance diagram for DDS::PublisherListener:



Additional Inherited Members

6.4.1 Detailed Description

The **PublisherListener** (p. 110) provides asynchronous notification of **Publisher** (p. 57) events.

This listener can be installed during **Publisher** (p. 57) creation DomainParticipant_create_publisher(), as well as by calling Publisher_set_listener().

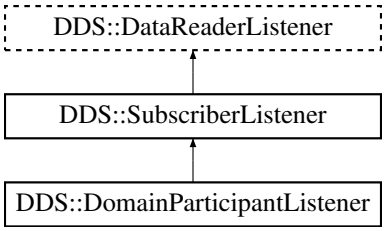
Note

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** (p. 44).

6.5 DDS::SubscriberListener Class Reference

The **SubscriberListener** (p. 111) provides asynchronous notification of **Subscriber** (p. 65) events.

Inheritance diagram for DDS::SubscriberListener:



Public Member Functions

- virtual void **on_data_on_readers** (**Subscriber** ∗)

6.5.1 Detailed Description

The **SubscriberListener** (p. 111) provides asynchronous notification of **Subscriber** (p. 65) events.

This listener can be installed during **Subscriber** (p. 65) 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** (p. 44).

6.5.2 Member Function Documentation

6.5.2.1 virtual void DDS::SubscriberListener::on_data_on_readers (Subscriber *) [inline], [virtual]

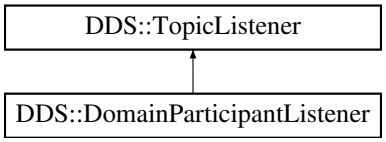
on_data_on_readers() (p. 112) is called when the CoreDX DDS middleware detects that data or data state information arrives at any **DataReader** (p. 26) contained within the **Subscriber** (p. 65).

The **status** argument provides a snapshot of the status at the time the listener was invoked.

6.6 DDS::TopicListener Class Reference

The **TopicListener** (p. 112) provides asynchronous notification of **Topic** (p. 69) events.

Inheritance diagram for DDS::TopicListener:



Public Member Functions

- virtual void **on_inconsistent_topic** (**Topic** *, const InconsistentTopicStatus &)

6.6.1 Detailed Description

The **TopicListener** (p. 112) provides asynchronous notification of **Topic** (p. 69) events.

This listener can be installed during **Topic** (p. 69) 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** (p. 44).

6.6.2 Member Function Documentation

6.6.2.1 virtual void DDS::TopicListener::on_inconsistent_topic (Topic *, const InconsistentTopicStatus &) [inline], [virtual]

on_inconsistent_topic() (p. 112) is called when the CoreDX DDS middleware detects that another topic exists with different (inconsistent) characteristics.

The **status** argument provides a snapshot of the status at the time the listener was invoked.

Chapter 7

DDS Status Structures

These data types are used to communicate status information in the Listener and get_XYZ_status() methods.

7.1 InconsistentTopicStatus Struct Reference

Status related to the on_inconsistent_topic listener methods of the DDS_TopicListener structure.

Public Attributes

- int **total_count**

  Cummulative count of the discovered Topics having a matching name and inconsistent characteristics.
- int **total_count_change**

  *Change in **total_count** since the last time the listener was called or status was read.*
- char **last_inconsistent_topic_msg** [256]

  Description of the most recently seen inconsistency.

7.1.1 Detailed Description

Status related to the on_inconsistent_topic listener methods of the DDS_TopicListener structure.

7.2 LivelinessChangedStatus Struct Reference

Status related to the on_liveliness_changed listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.

Public Attributes

- int **alive_count**

  The number of 'active' DataWriters matched to this DataReader.
- int **not_alive_count**

  The number of 'not-alive' DataWriters matched to this DataReader.
- int **alive_count_change**

*Change in **alive_count** since the last time the listener was called or status was read.*

- int **not_alive_count_change**

  *Change in **not_alive_count** since the last time the listener was called or status was read.*

- **DDS_InstanceHandle_t last_publication_handle**

  Handle identifying the most recent DataWriter whose liveliness changed.

7.2.1 Detailed Description

Status related to the on_liveliness_changed listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.

7.3 LivelinessLostStatus Struct Reference

Status related to the on_liveliness_lost listener methods of the DDS_DataWriter, DDS_Publisher, and DDS_DomainParticipant structures.

Public Attributes

- int **total_count**

  Cummulative number of times that an 'alive' DataWriter became not alive.

- int **total_count_change**

  *Change in **total_count** since the last time the listener was called or status was read.*

7.3.1 Detailed Description

Status related to the on_liveliness_lost listener methods of the DDS_DataWriter, DDS_Publisher, and DDS_DomainParticipant structures.

7.4 OfferedDeadlineMissedStatus Struct Reference

Status related to the on_offered_deadline_missed listener methods of the DDS_DataWriter, DDS_Publisher, and DDS_DomainParticipant structures.

Public Attributes

- int **total_count**

  Cummulative count of the number of deadlines missed by this DataWriter.

- int **total_count_change**

  *Change in **total_count** since the last time the listener was called or status was read.*

- **DDS_InstanceHandle_t last_instance_handle**

  Handle identifying the most recent instance whose deadline was missed.

7.4.1 Detailed Description

Status related to the on_offered_deadline_missed listener methods of the DDS_DataWriter, DDS_Publisher, and DDS_DomainParticipant structures.

7.5 OfferedIncompatibleQosStatus Struct Reference

Status related to the on_offered_incompatible_qos listener methods of the DDS_DataWriter, DDS_Publisher, and DDS_DomainParticipant structures.

Public Attributes

- int **total_count**

  Cummulative count of the number of DataWriters discovered having matching Topic and incompatible QoS.
- int **total_count_change**

  *Change in **total_count** since the last time the listener was called or status was read.*
- DDS_QosPolicyId_t **last_policy_id**

  Id of the most recent requested incompatible QoS policy.
- DDS_QosPolicyCountSeq **policies**

  A list of QoS policies and the total number of times each QoS policy was found to be incompatible.

7.5.1 Detailed Description

Status related to the on_offered_incompatible_qos listener methods of the DDS_DataWriter, DDS_Publisher, and DDS_DomainParticipant structures.

7.6 PublicationMatchedStatus Struct Reference

Status related to the on_publication_matched listener methods of the DDS_DataWriter, DDS_Publisher, and DDS_DomainParticipant structures.

Public Attributes

- int **total_count**

  Cummulative count of the number of times this DataWriter has discovered a matching DataReader.
- int **total_count_change**

  *Change in **total_count** since the last time the listener was called or status was read.*
- int **current_count**

  The current number of DataReaders matched to the DataWriter.
- int **current_count_change**

  *Change in **current_count** since the last time the listener was called or status was read.*

7.6.1 Detailed Description

Status related to the on_publication_matched listener methods of the DDS_DataWriter, DDS_Publisher, and DDS_DomainParticipant structures.

7.7 RequestedDeadlineMissedStatus Struct Reference

Status related to the on_requested_deadline_missed listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.

Public Attributes

- int **total_count**

  Cummulative count of the number of detected deadline misses for any instance read by the DataReader.
- int **total_count_change**

  *Change in **total_count** since the last time the listener was called or status was read.*
- **DDS_InstanceHandle_t last_instance_handle**

  Handle identifying the most recent instance whose deadline was missed.

7.7.1 Detailed Description

Status related to the on_requested_deadline_missed listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.

7.8 RequestedIncompatibleQosStatus Struct Reference

Status related to the on_requested_incompatible_qos listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.

Public Attributes

- int **total_count**

  Cummulative count of the number of discovered DataReaders having matching Topic and incompatible QoS.
- int **total_count_change**

  *Change in **total_count** since the last time the listener was called or status was read.*
- DDS_QosPolicyId_t **last_policy_id**

  Handle identifying the most recent QoS policy detected to be incompatible.
- DDS_QosPolicyCountSeq **policies**

  A list of QoS policies and the total number of times each QoS policy was found to be incompatible.

7.8.1 Detailed Description

Status related to the on_requested_incompatible_qos listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.

7.9 SampleLostStatus Struct Reference

Status related to the on_sample_lost listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.

Public Attributes

- int **total_count**

  Cummulative count of all samples lost under the Topic.
- int **total_count_change**

  *Change in **total_count** since the last time the listener was called or status was read.*

7.9.1 Detailed Description

Status related to the on_sample_lost listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.

7.10 SampleRejectedStatus Struct Reference

Status related to the on_sample_rejected listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.

Public Attributes

- int **total_count**

  Cummulative count of samples rejected by the DataReader.
- int **total_count_change**

  *Change in **total_count** since the last time the listener was called or status was read.*
- DDS_SampleRejectedStatusKind **last_reason**

  The reason for rejecting the most recently rejected sample.
- **DDS_InstanceHandle_t last_instance_handle**

  The handle of the instance associated with the most recently rejected sample.

7.10.1 Detailed Description

Status related to the on_sample_rejected listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.

7.11 SubscriptionMatchedStatus Struct Reference

Status related to the on_subscription_matched listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.

Public Attributes

- int **total_count**

  Cummulative count of the number of times this DataReader has discovered a matching DataWriter.
- int **total_count_change**

  *Change in **total_count** since the last time the listener was called or status was read.*
- int **current_count**

The current number of DataWriters matched to the DataReader.

- int **current_count_change**

  *Change in **current_count** since the last time the listener was called or status was read.*

7.11.1 Detailed Description

Status related to the on_subscription_matched listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.

Chapter 8

DDS Samples

The data types documented in this section are currently used only by the RPC API.

8.1 DDS::LoanedSamples< T > Class Template Reference

Holds a collection of sample data and related meta-data.

Public Types

- typedef **SampleIterator**< T, false > **iterator**
- typedef **SampleIterator**< T, true > **const_iterator**
- typedef dds_type_traits< T >::SampleIteratorValueType **value_type**
- typedef dds_type_traits< T >::ConstSampleIteratorValueType **const_value_type**
- typedef std::ptrdiff_t **difference_type**

Public Member Functions

- **LoanedSamples** ()

  Default constructor.
- ~**LoanedSamples** () throw ()

  Destructor.
- void **release** (DataReader &reader_ptr, Seq &**data_seq**, **DDS::SampleInfoSeq** &**info_seq**)

  Transfers the loan to the caller.
- **LoanedSamples** (LoanMemento loan_memento) throw ()

  Construct from a memento. Provides for efficient copy construction.
- **LoanedSamples** & **operator=** (LoanMemento loan_memento) throw ()

  Assignment operator that preserves loan ownership semantics.
- Seq & **data_seq** ()

  Access the contained data sequence.
- **DDS::SampleInfoSeq** & **info_seq** ()

  Access the contained sample info sequence.
- const Seq & **data_seq** () const

  Access the contained data sequence as const.

- const **DDS::SampleInfoSeq** & **info_seq** () const

  Access the contained sample info sequence as const.
- **value_type operator[]** (size_t index)

  Access a specific element of the contained data sequence. The returned value will be of type SampleRef<T>.
- **const_value_type operator[]** (size_t index) const

  Access a specific element of the contained data sequence as const The returned value will be of type SampleRef<const T>.
- int **length** () const

  Access the length of the contained data sequence.
- int **size** () const

  Access the length of the contained data sequence.
- void **return_loan** ()

  *Return the loaned data to the originating **DataReader** (p. 26).*
- void **swap** (**LoanedSamples** &that) throw ()

  Swap contents and responsibility between this and 'that'.
- **SharedSamples**< T > **to_shared** () throw ()

  *Convert the LoanedSample sequences into a **SharedSamples** (p. 126) object.*
- **iterator begin** ()

  *Obtain a **SampleIterator** (p. 123) object that refers to the beginning of the data sequence.*
- **iterator end** ()

  *Obtain a **SampleIterator** (p. 123) object that refers to the end of the data sequence.*
- **const_iterator begin** () const

  *Obtain a const **SampleIterator** (p. 123) object that refers to the beginning of the data sequence.*
- **const_iterator end** () const

  *Obtain a const **SampleIterator** (p. 123) object that refers to the end of the data sequence.*

Static Public Member Functions

- static **LoanedSamples**< T > **move_construct_from_loans** (DataReader reader, Seq &**data_seq**, **DDS::SampleInfoSeq** &**info_seq**)

  Construct from loaned sample and sample info sequences.

Friends

- **LoanedSamples move** (**LoanedSamples** &) throw ()

  move operator

8.1.1 Detailed Description

template<typename T>
class DDS::LoanedSamples< T >

Holds a collection of sample data and related meta-data.

The **LoanedSamples** (p. 119) object holds the loaned data and meta-data returned from a **DataReader::read()** (p. 32) operation. The object manages the 'loan' by automatically calling **return_loan()** (p. 122) on the source **DataReader** (p. 26) when the object is destroyed.

8.1.2 Member Typedef Documentation

8.1.2.1 template<typename T> typedef SampleIterator<T, true> DDS::LoanedSamples< T >::const_iterator

const_iterator typedef

8.1.2.2 template<typename T> typedef dds_type_traits<T>::ConstSampleIteratorValueType DDS::LoanedSamples< T >::const_value_type

const_value_type typedef. Is SampleRef<const T>

8.1.2.3 template<typename T> typedef std::ptrdiff_t DDS::LoanedSamples< T >::difference_type

difference_type typedef

8.1.2.4 template<typename T> typedef SampleIterator<T, false> DDS::LoanedSamples< T >::iterator

iterator typedef

8.1.2.5 template<typename T> typedef dds_type_traits<T>::SampleIteratorValueType DDS::LoanedSamples< T >::value_type

value_type typedef. Is SampleRef<T>

8.1.3 Member Function Documentation

8.1.3.1 template<typename T> static LoanedSamples<T> DDS::LoanedSamples< T >::move_construct_from_loans (DataReader *reader,* Seq & *data_seq,* DDS::SampleInfoSeq & *info_seq*) [static]

Construct from loaned sample and sample info sequences.

Takes over responbility of the loan. Will return when destroyed or explicitly via '**return_loan()** (p. 122)', or by transfering the responsibility via '**release()** (p. 121)'.

See also

return_loan() (p. 122)
release() (p. 121)

8.1.3.2 template<typename T> void DDS::LoanedSamples< T >::release (DataReader & *reader_ptr,* Seq & *data_seq,* DDS::SampleInfoSeq & *info_seq*)

Transfers the loan to the caller.

The **LoanedSamples** (p. 119) instance will no longer manage the loaned data. The caller is responsible for ensuring **DataReader::return_loan()** (p. 34) is invoked as required.

8.1.3.3 template<typename T> void DDS::LoanedSamples< T >::return_loan ( )

Return the loaned data to the originating **DataReader** (p. 26).

After this call, the **LoanedSamples** (p. 119) object no longer manages the data or sample info sequences.

8.1.3.4 template<typename T> SharedSamples<T> DDS::LoanedSamples< T >::to_shared ( ) throw)

Convert the LoanedSample sequences into a **SharedSamples** (p. 126) object.

The **SharedSamples** (p. 126) object holds a reference to the data and sample info sequences.

See also

SharedSamples (p. 126)

8.2 DDS::Sample< T > Class Template Reference

Sample (p. 122) holds the data and related meta information.

Public Member Functions

- **Sample** ()

  Default constructor. Empty data.
- **Sample** (const **Sample**< T > &other)

  Copy constructor. Duplicates data.
- **Sample** (**SampleRef**< T > sample_ref)

  Copy from Ref constructor. Duplicates data.
- **Sample** (const T &**data**, const DDS::SampleInfo &**info**)

  Constructor. Copies data.
- ~**Sample** ()

  Destructor. Clears data.
- **DDS::SampleIdentity_t identity** () const

  Access the sample's identity. Currently, the 'identity' is populated only by some RPC 'request' methods.
- **DDS::SampleIdentity_t related_identity** () const

  Access the sample's relatedIdentity. Currently, the 'relatedIdentity' is populated only if the sample originates from an RPC entity.
- void **swap** (**Sample** &other)

  Swap the contents of this with 'other'.
- T & **data** ()

  Access the data.
- DDS::SampleInfo & **info** ()

  Access the sample info.
- const T & **data** () const

  Const access to the data.
- const DDS::SampleInfo & **info** () const

  Const access to the sample info.
- void **set_data** (const T &t)

Set the data. Makes a copy.

- void **set_data** (const T *t)

  Set the data. Makes a copy.

- void **set_info** (const DDS::SampleInfo &i)

  Set the sample info. Makes a copy.

- void **set_info** (const DDS::SampleInfo *i)

  Set the sample info. Makes a copy.

- **Sample** & **operator=** (**Sample**< T > sample)

  Assignment operator.

- **Sample** & **operator=** (**SampleRef**< T > sample_ref)

  Assignment operator.

- **operator DataReference** ()

  Cast to a data reference.

- **operator ConstDataReference** () const

  Cast to a const data reference.

Friends

- class **DDS::DataReader**
- class **DDS::DataWriter**
- template<typename TReq , typename TRep >
  class **DDS::rpc::Requester**
- template<typename TReq , typename TRep >
  class **DDS::rpc::Replier**

8.2.1 Detailed Description

template<typename T>
class DDS::Sample< T >

Sample (p. 122) holds the data and related meta information.

Sample (p. 122) will allocate space for the sample data, and maintain that memory. The sample also includes SampleInfo which will be populated if the **Sample** (p. 122) instance is provided by a read() or similar operation of a DDS or RPC entity.

8.3 DDS::SampleIterator< T, IsConst > Class Template Reference

Provides a means to iterate over a collection of Samples.

Public Types

- typedef details::if_< IsConst, typename dds_type_traits< T >::ConstSampleIteratorValueType, typename dds_type_traits< T >::SampleIteratorValueType >::type **value_type**
- typedef std::ptrdiff_t **difference_type**

Public Member Functions

- **SampleIterator** ()

  Default constructor.
- **SampleIterator** (Seq &seq, InfoSeq &info_seq, int position=0)

  Construct from data and info sequences.
- **SampleIterator** (const **SampleIterator**< T, false > &si)

  Copy constructor.
- **SampleIterator** & **operator=** (const **SampleIterator**< T, false > &si)

  Assignment operator.
- **value_type operator**∗ () const

  Dereference operator. The returned object will be of SampleRef<T>
- **value_type operator->** () const

  Pointer operator The returned object will be of SampleRef<T>
- **value_type operator[]** (**difference_type** offset) const

  Array subscript The returned object will be of SampleRef<T>
- **SampleIterator** & **operator++** ()

  Prefix increment operator.
- **SampleIterator operator++** (int)

  Postfix increment operator.
- **SampleIterator** & **operator--** ()

  Prefix decrement operator.
- **SampleIterator operator--** (int)

  Postfix decrement operator.
- **SampleIterator** & **operator-=** (**difference_type** i)

  In-place subtraction.
- **SampleIterator** & **operator+=** (**difference_type** i)

  In-place addition.
- **SampleIterator operator-** (**difference_type** i) const

  Subtraction operator.
- **SampleIterator operator+** (**difference_type** i) const

  Addition operator.
- **difference_type operator-** (const **SampleIterator** &s2) const

  Subtraction operator.
- bool **operator**< (const **SampleIterator** &s2) const

  Less than comparison.
- bool **operator**> (const **SampleIterator** &s2) const

  Greater than comparison.
- bool **operator<=** (const **SampleIterator** &s2) const

  Less than or equal comparison.
- bool **operator>=** (const **SampleIterator** &s2) const

  Greater than or equal comparison.
- bool **operator==** (const **SampleIterator** &s2) const

  Equality comparison.
- bool **operator!=** (const **SampleIterator** &s2) const

  Inequality comparison.
- bool **is_end** () const

  Test if iterator is at the end of the data.

8.3.1 Detailed Description

template<typename T, bool IsConst>
class DDS::SampleIterator< T, IsConst >

Provides a means to iterate over a collection of Samples.

See also

LoanedSamples (p. 119)
SharedSamples (p. 126)

8.3.2 Member Typedef Documentation

8.3.2.1 template<typename T, bool IsConst> typedef std::ptrdiff_t DDS::SampleIterator< T, IsConst >::difference_type

difference_type typedef

8.3.2.2 template<typename T, bool IsConst> typedef details::if_<IsConst, typename dds_type_traits<T>::ConstSampleIteratorValueType, typename dds_type_traits<T>::SampleIteratorValueType>::type DDS::SampleIterator< T, IsConst >::value_type

value_type typedef. Is SampleRef<T> or SampleRef<const T>

8.4 DDS::SampleRef< T > Class Template Reference

SampleRef (p. 125) holds a reference to data and related meta information.

Public Member Functions

- **SampleRef** ()

  Default constructor. Null data.
- **SampleRef** (T *__data__, DDS::SampleInfo *__info__)

  Construct with provided data and info pointers.
- **SampleRef** (T &__data__, DDS::SampleInfo &__info__)

  Construct with provided data and info references.
- **SampleRef** (**Sample**< T > &sample)

  Construct with data and info references to the provided Sample<T>.
- **SampleRef** & **operator=** (**Sample**< T > &sample)

  Assignment operator, data and info refer to the data and info in the provided Sample<T>.
- T & **data** () const

  Access the data.
- DDS::SampleInfo & **info** () const

  Access the sample info.
- **operator T &** () const

  Cast to a T reference.
- void **set_data** (T *__data__)

Assign the data reference.

- void **set_data** (T &**data**)

  Assign the data reference.

- void **set_info** (DDS::SampleInfo ∗**info**)

  Assign the sample info reference.

- void **set_info** (DDS::SampleInfo &**info**)

  Assign the sample info reference.

- bool **is_nil_data** () const

  Test if the data is NULL.

- bool **is_nil_info** () const

  Test if the sample info is NULL.

- void **swap** (**SampleRef** &other) throw ()

  Swap the data and info references from this to other.

- **DDS::SampleIdentity_t identity** () const

  Access the sample identity. Currently, the 'identity' is populated only by some RPC 'request' methods.

- **DDS::SampleIdentity_t related_identity** () const

  Access the sample's relatedIdentity. Currently, the 'relatedIdentity' is populated only if the sample originates from an RPC entity.

Friends

- class **DDS::DataReader**
- class **DDS::DataWriter**
- template<typename TReq , typename TRep >
  class **DDS::rpc::Requester**
- template<typename TReq , typename TRep >
  class **DDS::rpc::Replier**

8.4.1 Detailed Description

template<typename T>
class DDS::SampleRef< T >

SampleRef (p. 125) holds a reference to data and related meta information.

SampleRef (p. 125) will hold a pointer to sample data. **SampleRef** (p. 125) does not clear or destroy the sample data. Further the sample data must remain valid during the lifetime of the **SampleRef** (p. 125).

The **SampleRef** (p. 125) also includes SampleInfo which will be populated if the **SampleRef** (p. 125) instance is provided by a read() or similar operation of a DDS or RPC entity.

8.5 DDS::SharedSamples< T > Class Template Reference

Holds a reference to a collection of sample data and related meta-data.

Public Types

- typedef **SampleIterator**< T, false > **iterator**
- typedef **SampleIterator**< T, true > **const_iterator**
- typedef dds_type_traits< T >::SampleIteratorValueType **value_type**
- typedef dds_type_traits< T >::ConstSampleIteratorValueType **const_value_type**
- typedef std::ptrdiff_t **difference_type**

Public Member Functions

- **SharedSamples** ()

  Default constructor.
- **SharedSamples** (const **SharedSamples** &)

  Copy constructor.
- ~**SharedSamples** () throw ()

  Destructor.
- **SharedSamples** & **operator=** (const **SharedSamples** &)

  Assignment operator.
- Seq & **data_seq** ()

  Access the contained data sequence.
- **DDS::SampleInfoSeq** & **info_seq** ()

  Access the contained sample info sequence.
- const Seq & **data_seq** () const

  Access the contained data sequence as const.
- const **DDS::SampleInfoSeq** & **info_seq** () const

  Access the contained sample info sequence as const.
- **value_type operator[]** (size_t index)

  Access a specific element of the data sequence The returned value will be of type SampleRef<T>
- **const_value_type operator[]** (size_t index) const

  Access a specific element of the data sequence as const The returned value will be of type SampleRef<const T>
- int **length** () const

  Access the length of the contained data sequence.
- void **swap** (**SharedSamples** &that) throw ()

  Swap the contents of this with 'that'.
- **iterator begin** ()

  Obtain an iterator that refers to the beginning of the contained sequences.
- **iterator end** ()

  Obtain an iterator that refers to the end of the contained sequences.
- **const_iterator begin** () const

  Obtain a const iterator that refers to the beginning of the contained sequences.
- **const_iterator end** () const

  Obtain a const iterator that refers to the end of the contained sequences.

8.5.1 Detailed Description

template<typename T>
class DDS::SharedSamples< T >

Holds a reference to a collection of sample data and related meta-data.

This object provides a counted reference to the data, ensuring that while it exists, the data will continue to be valid. When the last **SharedSamples** (p. 126) instance is destroyed [the last reference is removed], then the original loan is ultimately returned as required.

8.5.2 Member Typedef Documentation

8.5.2.1 template<typename T> typedef SampleIterator<T, true> DDS::SharedSamples< T >::const_iterator

const_iterator typedef

8.5.2.2 template<typename T> typedef dds_type_traits<T>::ConstSampleIteratorValueType DDS::SharedSamples< T >::const_value_type

const_value_type typedef. Is SampleRef<const T>

8.5.2.3 template<typename T> typedef std::ptrdiff_t DDS::SharedSamples< T >::difference_type

difference_type typedef

8.5.2.4 template<typename T> typedef SampleIterator<T, false> DDS::SharedSamples< T >::iterator

iterator typedef

8.5.2.5 template<typename T> typedef dds_type_traits<T>::SampleIteratorValueType DDS::SharedSamples< T >::value_type

value_type typedef. Is SampleRef<T>

8.6 DDS::WriteSample< T > Class Template Reference

Holds sample data intended for transmission.

Public Member Functions

- **WriteSample** ()

  Default constructor.
- **WriteSample** (const **WriteSample**< T > &other)

  Copy constructor. Allocates a new instance, and copies from 'other'.
- **WriteSample** (const T &**data**)

  Constructor. Allocates a new instance, and copies from 'data'.

- **WriteSample** (const **WriteSampleRef**< T > &wsref)

  Constructor. Allocates a new instance, and copies from the data referenced in 'wsref'.
- ~**WriteSample** ()

  Destructor. Deletes any contained data.
- **WriteSample** & **operator=** (const **WriteSampleRef**< T > &wsref)

  *Assignment operator (from **WriteSampleRef** (p. 129))*
- **WriteSample** & **operator=** (const **WriteSample**< T > &ws)

  Assignment operator.
- const T & **data** () const

  Const access to the contained data instance.
- T & **data** ()

  Access to the contained data instance. May allocate a new 'T' if hasn't yet.
- void **set_data** (const T &**data**)

  Assign the data. Copies 'data' into internal state.
- void **set_identity** (const **DDS::SampleIdentity_t** &id)

  *Assign the identity of the **Sample** (p. 122).*
- void **swap** (**WriteSample** &other) throw ()

  Swap the contents of this with 'other'.
- **DDS::SampleIdentity_t identity** () const

  *Access the identity of this **Sample** (p. 122).*

Friends

- class **DDS::DataReader**
- class **DDS::DataWriter**
- template<typename TReq , typename TRep >
  class **DDS::rpc::Requester**
- template<typename TReq , typename TRep >
  class **DDS::rpc::Replier**

8.6.1 Detailed Description

template<typename T>
class DDS::WriteSample< T >

Holds sample data intended for transmission.

8.7 DDS::WriteSampleRef< T > Class Template Reference

Holds a reference to sample data intended for transmission.

Public Member Functions

- **WriteSampleRef** ()

  Default constructor.
- **WriteSampleRef** (T &**data**)

Constructor. Takes a reference to 'data'.

- **WriteSampleRef** (**WriteSample**< T > &ws)

  Constructor. Takes a reference to 'ws' data.
- **WriteSampleRef** (const **WriteSampleRef**< T > &wsref)

  Constructor. Takes a reference to 'wsref' data.
- **WriteSampleRef** & **operator=** (**WriteSample**< T > &ws)

  Assignment operator. Takes a reference to ws data.
- **WriteSampleRef** & **operator=** (const **WriteSampleRef**< T > &wsref)

  Assignment operator. Takes a reference to wsref data.
- const T & **data** () const

  Const access to data.
- T & **data** ()

  Access to data.
- void **set_data** (T &**data**)

  Assign data, takes a reference to 'data'.
- void **set_identity** (const **DDS::SampleIdentity_t** &id)

  *Assign the **Sample** (p. 122) identity.*
- void **swap** (**WriteSampleRef** &other) throw ()

  Swap the contents of this with 'other'.
- **DDS::SampleIdentity_t identity** () const

  *Access the **Sample** (p. 122) identity.*

Friends

- class **DDS::DataReader**
- class **DDS::DataWriter**
- template<typename TReq , typename TRep >
  class **DDS::rpc::Requester**
- template<typename TReq , typename TRep >
  class **DDS::rpc::Replier**

8.7.1 Detailed Description

template<typename T>
class DDS::WriteSampleRef< T >

Holds a reference to sample data intended for transmission.

Chapter 9

Builtin DDS Types

9.1 DDS::BuiltinTopicKey_t Struct Reference

Public Member Functions

- **BuiltinTopicKey_t** ()

Public Attributes

- int **value** [3]

9.1.1 Detailed Description

Holds the 'key' for a builtin topic

9.1.2 Constructor & Destructor Documentation

9.1.2.1 DDS::BuiltinTopicKey_t::BuiltinTopicKey_t ( )

Constructor, Copy Constructor, Destructor, Assignment operator

9.1.3 Member Data Documentation

9.1.3.1 int DDS::BuiltinTopicKey_t::value[3]

12 bytes of key data

9.2 DDS::Duration_t Struct Reference

The **Duration_t** (p. 131) structure contains data to define a time duration.

Public Member Functions

- **Duration_t** ()

9.2.1 Detailed Description

The **Duration_t** (p. 131) structure contains data to define a time duration.

9.2.2 Constructor & Destructor Documentation

9.2.2.1 DDS::Duration_t::Duration_t ( )

Constructor, Copy Constructor, Destructor, Assignment operator

9.3 DDS::GUID_t Struct Reference

Public Member Functions

- **GUID_t** ()

Public Attributes

- unsigned char **value** [16]

9.3.1 Detailed Description

A complete GUID that uniquely identifies a **DDS** (p. 313) entity.

9.3.2 Constructor & Destructor Documentation

9.3.2.1 DDS::GUID_t::GUID_t ( )

Constructor, Copy Constructor, Destructor, Assignment operator

9.3.3 Member Data Documentation

9.3.3.1 unsigned char DDS::GUID_t::value[16]

the guid bytes

9.4 DDS::Property_t Struct Reference

A name-value pair property. The 'propagate' flag indicates if this property should be transfered through discovery.

Public Member Functions

- **Property_t** ()

Public Attributes

- char ∗ **name**

  the property name
- char ∗ **value**

  the property value
- unsigned char **propagate**

  propagate over discovery?

9.4.1 Detailed Description

A name-value pair property. The 'propagate' flag indicates if this property should be transfered through discovery.

9.4.2 Constructor & Destructor Documentation

9.4.2.1 DDS::Property_t::Property_t ( )

Constructor, Copy Constructor, Destructor, Assignment operator

9.5 DDS::SampleIdentity_t Struct Reference

Public Member Functions

- **SampleIdentity_t** ()

Public Attributes

- struct **DDS::GUID_t guid**

9.5.1 Detailed Description

Uniquely identifies a sample and its source entity.

9.5.2 Constructor & Destructor Documentation

9.5.2.1 DDS::SampleIdentity_t::SampleIdentity_t ( )

Constructor, Copy Constructor, Destructor, Assignment operator

9.5.3 Member Data Documentation

9.5.3.1 struct DDS::GUID_t DDS::SampleIdentity_t::guid

The source of the sample. The sequence number of the sample

9.6 DDS::SequenceNumber_t Struct Reference

Public Member Functions

- **SequenceNumber_t** ()

Public Attributes

- int **high**
- uint32_t **low**

9.6.1 Detailed Description

A sequence number that uniquely identifies a sample within the scope of a writer cache.

9.6.2 Constructor & Destructor Documentation

9.6.2.1 DDS::SequenceNumber_t::SequenceNumber_t ( )

Constructor, Copy Constructor, Destructor, Assignment operator

9.6.3 Member Data Documentation

9.6.3.1 int DDS::SequenceNumber_t::high

The 'high' 32 bits of the sequence number

9.6.3.2 uint32_t DDS::SequenceNumber_t::low

The 'low' 32 bits of the sequence number

9.7 DDS::Time_t Struct Reference

The **Time_t** (p. 134) structure contains data to define a time.

9.7.1 Detailed Description

The **Time_t** (p. 134) structure contains data to define a time.

9.8 DDS_Time_t Struct Reference

Public Attributes

- int **sec**
- uint32_t **nanosec**

9.8.1 Detailed Description

Defines a time

9.8.2 Member Data Documentation

9.8.2.1 uint32_t DDS_Time_t::nanosec

Time nanoseconds

9.8.2.2 int DDS_Time_t::sec

Time seconds

Chapter 10

DDS Discovery Types

10.1 DDS::DCPSParticipant Struct Reference

Public Member Functions

- **DCPSParticipant** ()

Public Attributes

- struct **DDS::BuiltinTopicKey_t key**

  key for this participant (the 12 byte prefix)
- struct **DDS::UserDataQosPolicy user_data**

  user_data associated with this participant
- struct **DDS::PropertyQosPolicy properties**

  key-value pairs associated with this participant
- char ∗ **entity_name**

  arbitrary string name assigned to participant

10.1.1 Detailed Description

The **DCPSParticipant** (p. 137) structure holds discovered Participant data.

10.1.2 Constructor & Destructor Documentation

10.1.2.1 DDS::DCPSParticipant::DCPSParticipant ( )

Constructor, Copy Constructor, Destructor, Assignment operator

10.2 DDS::DCPSPublication Struct Reference

Public Member Functions

- **DCPSPublication** ()

Public Attributes

- struct **DDS::BuiltinTopicKey_t participant_key**

  the prefix of containing participant
- struct **DDS::BuiltinTopicKey_t key**

  the entity id of this entity
- char ∗ **topic_name**

  the topic name
- char ∗ **type_name**

  type of the type
- struct **DDS::DurabilityQosPolicy durability**

  durability config of entity
- struct **DDS::DurabilityServiceQosPolicy durability_service**

  durability service config of this entity
- struct **DDS::DeadlineQosPolicy deadline**

  deadline config of this entity
- struct **DDS::LatencyBudgetQosPolicy latency_budget**

  latency budget config of this entity
- struct **DDS::LivelinessQosPolicy liveliness**

  liveliness config of this entity
- struct **DDS::ReliabilityQosPolicy reliability**

  reliability config of this entity
- struct **DDS::LifespanQosPolicy lifespan**

  lifespan config of this entity
- struct **DDS::UserDataQosPolicy user_data**

  user data config of this entity
- struct **DDS::OwnershipQosPolicy ownership**

  ownership config of this entity
- struct **DDS::OwnershipStrengthQosPolicy ownership_strength**

  ownership strength config of this entity
- struct **DDS::DestinationOrderQosPolicy destination_order**

  destination order config of this entity
- struct **DDS::PresentationQosPolicy presentation**

  presentation config of this entity
- struct **DDS::PartitionQosPolicy partition**

  partition config of this entity
- struct **DDS::TopicDataQosPolicy topic_data**

  topic data config of this entity
- struct **DDS::GroupDataQosPolicy group_data**

  group data config of this entity
- struct **DDS::DataRepresentationQosPolicy representation**

  data representation config of this entity

10.2.1 Detailed Description

The **DCPSPublication** (p. 137) structure holds discovered **DataWriter** (p. 37) data.

10.2.2 Constructor & Destructor Documentation

10.2.2.1 DDS::DCPSPublication::DCPSPublication ( )

Constructor, Copy Constructor, Destructor, Assignment operator

10.3 DDS::DCPSSubscription Struct Reference

Public Member Functions

- **DCPSSubscription** ()

Public Attributes

- struct **DDS::BuiltinTopicKey_t participant_key**

  containing participant guid prefix config of this entity
- struct **DDS::BuiltinTopicKey_t key**

  the entity id this entity
- char ∗ **topic_name**

  topic name config of this entity
- char ∗ **type_name**

  type name config of this entity
- struct **DDS::DurabilityQosPolicy durability**

  durability config of this entity
- struct **DDS::DeadlineQosPolicy deadline**

  deadline config of this entity
- struct **DDS::LatencyBudgetQosPolicy latency_budget**

  latency budget config of this entity
- struct **DDS::LivelinessQosPolicy liveliness**

  liveliness config of this entity
- struct **DDS::ReliabilityQosPolicy reliability**

  realiability config of this entity
- struct **DDS::OwnershipQosPolicy ownership**

  ownership config of this entity
- struct **DDS::DestinationOrderQosPolicy destination_order**

  destination order config of this entity
- struct **DDS::UserDataQosPolicy user_data**

  user data config of this entity
- struct **DDS::TimeBasedFilterQosPolicy time_based_filter**

  time based filter config of this entity
- struct **DDS::PresentationQosPolicy presentation**

  presentation config of this entity
- struct **DDS::PartitionQosPolicy partition**

  partition config of this entity
- struct **DDS::TopicDataQosPolicy topic_data**

  topic data config of this entity

- struct **DDS::GroupDataQosPolicy group_data**

  group data config of this entity
- struct **DDS::DataRepresentationQosPolicy representation**

  representation config of this entity
- struct **DDS::TypeConsistencyEnforcementQosPolicy type_consistency**

  type consistency config of this entity
- char ∗ **entity_name**

  entity name config of this entity
- struct **DDS::RpcQosPolicy rpc**

  rpc config of this entity

10.3.1 Detailed Description

The **DCPSSubscription** (p. 139) structure holds discovered **DataReader** (p. 26) data.

10.3.2 Constructor & Destructor Documentation

10.3.2.1 DDS::DCPSSubscription::DCPSSubscription ( )

Constructor, Copy Constructor, Destructor, Assignment operator

Chapter 11

Supporting Types

11.1 CacheStatistics Struct Reference

Encapsulates statistics available from a DataReader or DataWriter.

Public Attributes

- int32_t **instance_count**
- int32_t **instance_alive_count**
- int32_t **instance_disposed_count**
- int32_t **instance_no_writers_count**
- int32_t **instance_new_count**
- int32_t **sample_count**
- int32_t **sample_read_count**
- uint32_t **state_flags**

11.1.1 Detailed Description

Encapsulates statistics available from a DataReader or DataWriter.

Queries the current state of the sample and instance cache. A snapshot in time of the current state is written to the provided **DDS_CacheStatistics** (p. 141) parameter. The statistics include sample and instance counts.

Return values

DDS_RETCODE_OK	on success
DDS_RETCODE_NO_DATA	if unable to get the data

See also

DDS_DataReader_get_cache_stats()
DDS_DataWriter_get_cache_stats()

11.1.2 Member Data Documentation

11.1.2.1 int32_t CacheStatistics::instance_alive_count

number of alive instances

11.1.2.2 int32_t CacheStatistics::instance_count

total number of instances

11.1.2.3 int32_t CacheStatistics::instance_disposed_count

number of disposed instances

11.1.2.4 int32_t CacheStatistics::instance_new_count

number of 'new' (view state) instances

11.1.2.5 int32_t CacheStatistics::instance_no_writers_count

number of no_writer instances

11.1.2.6 int32_t CacheStatistics::sample_count

total number of samples

11.1.2.7 int32_t CacheStatistics::sample_read_count

number of 'read' samples

11.1.2.8 uint32_t CacheStatistics::state_flags

flags indicating internal state

11.2 Locator Struct Reference

Network address.

11.2.1 Detailed Description

Network address.

For a list of suppored LOCATOR_KINDs, refer to include/dds/dds.h

11.3 ParticipantLocator Struct Reference

Describes a the location and identity of a potential peer DomainParticipant.

Public Attributes

- unsigned int **participant_id**

  participant id start
- unsigned int **participant_id_max**

  participant id max
- **Locator participant_locator**

  locator identifying peer participant

11.3.1 Detailed Description

Describes a the location and identity of a potential peer DomainParticipant.

11.4 QosPolicyCount Struct Reference

Holds a DDS_QosPolicyId_t value and a counter to indicate the number of events associated with that PolicyId.

Public Attributes

- DDS_QosPolicyId_t **policy_id**
- int **count**

11.4.1 Detailed Description

Holds a DDS_QosPolicyId_t value and a counter to indicate the number of events associated with that PolicyId.

11.4.2 Member Data Documentation

11.4.2.1 int QosPolicyCount::count

Counter associated with this PolicyId.

11.4.2.2 DDS_QosPolicyId_t QosPolicyCount::policy_id

QosPolicyCount value.

11.5 SampleInfo Struct Reference

The SampleInfo structure contains information associated with each sample.

Public Attributes

- DDS_SampleStateKind **sample_state**

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

- DDS_ViewStateKind **view_state**

  Associated instance has/has not been seen before.
- DDS_InstanceStateKind **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.
- **DDS_Time_t source_timestamp**

  The time provided by the DataWriter when the sample was written.
- **DDS_Time_t reception_timestamp**

  The time the sample was received by the DataReader.
- **DDS_InstanceHandle_t instance_handle**

  The handle that locally identifies the associated instance.
- **DDS_InstanceHandle_t publication_handle**

  The local handle of the source DataWriter.
- 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.)
- 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.)
- int **sample_rank**

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

  The generation difference of this sample and the most recent sample in the collection.
- int **absolute_generation_rank**

  The generation difference between this sample and the most recent sample.
- unsigned char **valid_data**

  Set to true (non-zero) if the associated DataSample contains data.

11.5.1 Detailed Description

The SampleInfo structure contains information associated with each sample.

The C++ SampleInfo structure contains information associated with each sample.

11.5.2 Member Data Documentation

11.5.2.1 int SampleInfo::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.

11.5.2.2 int SampleInfo::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.

11.5.2.3 DDS_InstanceStateKind SampleInfo::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) ...
```

11.5.2.4 DDS_SampleStateKind SampleInfo::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.

11.5.2.5 unsigned char SampleInfo::valid_data

Set to true (non-zero) 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).

11.5.2.6 DDS_ViewStateKind SampleInfo::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

11.6 DDS::sequence< T > Class Template Reference

Sequence data type.

Public Member Functions

- **sequence** ()

  construct an empty sequence
- **sequence** (const **sequence** &x)

  copy constructor Copies sequence contents.
- ∼**sequence** ()

  destroy the sequence and release resources. Destroys sequence contents.
- **sequence** & **operator=** (const **sequence** &x)

  assignment operator Copies seqeuence contents.
- uint32_t **size** () const

  Returns the current size (number of elements) of the sequence.
- uint32_t **length** () const

  *alias for **size()*** (p. 146)
- uint32_t **capacity** () const

  Returns the current capacity (number of elements) of the sequence. The equation "size <= capacity" is always true.
- bool **reserve** (uint32_t n)

  Set the capacity of the sequence.
- bool **resize** (uint32_t n)

  Set the 'capacity' and 'size' of the sequence.
- void **maximum** (uint32_t n)

  set the 'maximum' length of the sequence
- uint32_t **maximum** () const

  set the 'maximum' length of the sequence
- bool **empty** () const

  test if the sequence is empty
- void **shrink_to_fit** ()

  reduce 'capacity' to match 'size'
- void **assign** (T ∗**buffer**, uint32_t n)

  Caller provides an array of elements of size 'n'. The Sequence takes ownership of the memory. Note, 'buffer' must be allocated via new[].
- T ∗ **release** ()

  Release current sequence contents, caller takes ownership of the memory. If successful, the sequence is empty after this operation.
- T & **operator[]** (uint32_t n)
- const T & **operator[]** (uint32_t n) const
- T & **at** (uint32_t n)
- const T & **at** (uint32_t n) const
- T & **front** ()
- const T & **front** () const
- T & **back** ()
- const T & **back** () const
- const T ∗ **buffer** () const
- void **push_back** (const T &v)
- void **pop_back** ()
- void **swap** (**sequence** &x)
- void **clear** ()

11.6.1 Detailed Description

template<class T>
class DDS::sequence< T >

Sequence data type.

Provides an implementation of the Sequence data type. This data type is used in a few places throughout the DDS API.

It is most prevelant in the **DataReader** (p. 26) API where it is used to return a collection of samples and SampleInfo items back to the caller.

It is also used in a few QoS Policies.

11.6.2 Member Function Documentation

11.6.2.1 template<class T > T & DDS::sequence< T >::at (uint32_t *n*)

access an element

11.6.2.2 template<class T > const T & DDS::sequence< T >::at (uint32_t *n*) const

access an element

11.6.2.3 template<class T > T & DDS::sequence< T >::back ( )

access the last element

11.6.2.4 template<class T > const T & DDS::sequence< T >::back ( ) const

access the last element

11.6.2.5 template<class T > const T * DDS::sequence< T >::buffer ( ) const

get access to underlying buffer

11.6.2.6 template<class T > void DDS::sequence< T >::clear ( )

clear the contents of this sequence

11.6.2.7 template<class T > T & DDS::sequence< T >::front ( )

access the first element

11.6.2.8 template<class T > const T & DDS::sequence< T >::front ( ) const

access the first element

11.6.2.9 template<class T > T & DDS::sequence< T >::operator[] (uint32_t *n*)

access an element

11.6.2.10 template<class T > const T & DDS::sequence< T >::operator[] (uint32_t *n*) const

access an element

11.6.2.11 template<class T > void DDS::sequence< T >::pop_back ( )

reduce 'size' by one

11.6.2.12 template<class T> void DDS::sequence< T >::push_back (const T & *v*)

add an element after the last Increments 'size' (and possibly 'capacity') if requried.

11.6.2.13 template<class T > bool DDS::sequence< T >::reserve (uint32_t *n*)

Set the capacity of the sequence.

If n is larger than current capacity, then the capacity is increased to 'n'. If 'n' is smaller than current capacity, then the capacity is reduced to 'n'. If capacity is reduced below 'size', then 'size' will be reduced to match 'capacity'. In the case of a reduction in 'size', the elimitated elements will be destroyed.

11.6.2.14 template<class T > bool DDS::sequence< T >::resize (uint32_t *n*)

Set the 'capacity' and 'size' of the sequence.

If n is larger than current capacity, then the capacity is increased to 'n'. If 'n' is smaller than current capacity, then the capacity is reduced to 'n'. If capacity is reduced below 'size', then 'size' will be reduced to match 'capacity'. In the case of a reduction in 'size', the elimitated elements will be destroyed.

11.6.2.15 template<class T > void DDS::sequence< T >::swap (sequence< T > & *x*)

swap the contents of this sequence with that of another

Chapter 12

X-Types DynamicType API

12.1 DDS::AnnotationDescriptor Class Reference

A **AnnotationDescriptor** (p. 149) object comprises the state of an annotation as it is applied to some element.

Public Member Functions

- **AnnotationDescriptor** ()
- **ReturnCode_t get_value** (char ∗∗value, const char ∗key)

  This accesses an annotation property 'value' associated with the 'key' name.
- **ReturnCode_t get_all_value** (Parameters ∗value)

  Access the map of all key-value pairs.
- **ReturnCode_t set_value** (const char ∗key, const char ∗value)
- **ReturnCode_t copy_from** (const class **AnnotationDescriptor** ∗other)
- unsigned char **equals** (const class **AnnotationDescriptor** ∗other)
- unsigned char **is_consistent** ()

Public Attributes

- class **DynamicType** ∗ **type**

12.1.1 Detailed Description

A **AnnotationDescriptor** (p. 149) object comprises the state of an annotation as it is applied to some element.

12.1.2 Constructor & Destructor Documentation

12.1.2.1 DDS::AnnotationDescriptor::AnnotationDescriptor ( )

Constructor, Copy Constructor, Destructor, Assignment operator

12.1.3 Member Function Documentation

12.1.3.1 ReturnCode_t DDS::AnnotationDescriptor::copy_from (const class AnnotationDescriptor * *other*)

Not Yet Supported This operation is not yet implemented.

12.1.3.2 unsigned char DDS::AnnotationDescriptor::equals (const class AnnotationDescriptor * *other*)

Not Yet Supported This operation is not yet implemented.

12.1.3.3 ReturnCode_t DDS::AnnotationDescriptor::get_all_value (Parameters * *value*)

Access the map of all key-value pairs.

Return values

RETCODE_OK	on success, the 'value' parameters map is populated.

Not Yet Supported This operation is not yet implemented.

12.1.3.4 ReturnCode_t DDS::AnnotationDescriptor::get_value (char ** *value,* const char * *key*)

This accesses an annotation property 'value' associated with the 'key' name.

Every attribute value is represented as a string although annotation type members can have other types as well. A string representation of a data value is considered well formed if it would be a valid IDL literal of the corresponding type with the following qualifications:

- String and character literals shall not be surrounded by quotation characters (‘"’ or ‘'’)
- All expressions shall be fully evaluated such that no operators or other non-literal characters occur in the value. For example, "5" is a well-formed string representation of the integer quantity five, but "2 + ENUM_VALUE_THREE" is not.

  Not Yet Supported This operation is not yet implemented.

12.1.3.5 unsigned char DDS::AnnotationDescriptor::is_consistent ( )

Not Yet Supported This operation is not yet implemented.

12.1.3.6 ReturnCode_t DDS::AnnotationDescriptor::set_value (const char * *key,* const char * *value*)

Not Yet Supported This operation is not yet implemented.

12.1.4 Member Data Documentation

12.1.4.1 class DynamicType∗ DDS::AnnotationDescriptor::type

the annotation type

12.2 DDS::DynamicData Class Reference

A **DynamicData** (p. 151) object represents an individual data sample. It provides reflective getters and setters for the members of that sample.

Public Member Functions

- virtual const class **DynamicType** ∗ **get_type** (void)=0

  This provides the type that defines the values within this sample.
- virtual **ReturnCode_t get_descriptor** (class **MemberDescriptor** ∗value, const MemberId id)=0

  This provides access to the descriptor for each value in this object, identified by the member ID.
- virtual **ReturnCode_t set_descriptor** (const MemberId id, const class **MemberDescriptor** ∗value)=0

  Set the descriptor of the member specified by 'id'.
- virtual unsigned char **equals** (const class **DynamicData** ∗other)=0

  *Compare two **DynamicData** (p. 151) instances for equality.*
- virtual MemberId **get_member_id_by_name** (const char ∗name)=0

  Access the 'id' of a member matching 'name'.
- virtual MemberId **get_member_id_at_index** (const unsigned int index)=0

  Access the 'id' of a member at the specified index.
- virtual unsigned int **get_item_count** ()=0

  Access the "item count" of the DDS_DynamicData.
- virtual **ReturnCode_t get_name_by_id** (const char ∗∗name_str, const MemberId id)=0

  Support for map instances. Access the key name by 'id' (index).
- virtual **ReturnCode_t clear_all_values** ()=0

  Clears all values from the instance.
- virtual **ReturnCode_t clear_nonkey_values** ()=0

  Clear all non-key values from the instance.
- virtual **ReturnCode_t clear_value** (const MemberId id)=0

  Clear the specified value from the instance.
- virtual class **DynamicData** ∗ **loan_value** (const MemberId id)=0

  *The "loan" operations loan to the application a **DynamicData** (p. 151) object representing a value within this sample.*
- virtual **ReturnCode_t return_loaned_value** (const class **DynamicData** ∗value)=0

  Return a "loaned" data sample.
- virtual class **DynamicData** ∗ **clone** ()=0

  Create and return a new data sample with the same contents as this one.
- virtual **ReturnCode_t get_int32_value** (int ∗value, const MemberId id)=0

  Access a 32bit integer value, identified by 'id', in the sample.
- virtual **ReturnCode_t set_int32_value** (const MemberId id, const int value)=0

  Set a 32bit integer value, identified by 'id', in the sample.
- virtual **ReturnCode_t get_uint32_value** (unsigned int ∗value, const MemberId id)=0

Access an unsigned 32bit integer value, identified by 'id', in the sample.

- virtual **ReturnCode_t set_uint32_value** (const MemberId id, const unsigned int value)=0

  Set an unsigned 32bit integer value, identified by 'id', in the sample.
- virtual **ReturnCode_t get_int16_value** (short ∗value, const MemberId id)=0

  Access a 16bit integer value, identified by 'id', in the sample.
- virtual **ReturnCode_t set_int16_value** (const MemberId id, const short value)=0

  Set a 16bit integer value, identified by 'id', in the sample.
- virtual **ReturnCode_t get_uint16_value** (unsigned short ∗value, const MemberId id)=0

  Access an unsigned 16bit integer value, identified by 'id', in the sample.
- virtual **ReturnCode_t set_uint16_value** (const MemberId id, const unsigned short value)=0

  Set an unsigned 16bit integer value, identified by 'id', in the sample.
- virtual **ReturnCode_t get_int64_value** (int64_t ∗value, const MemberId id)=0

  Access a 64bit integer value, identified by 'id', in the sample.
- virtual **ReturnCode_t set_int64_value** (const MemberId id, const int64_t value)=0

  Set a 64bit integer value, identified by 'id', in the sample.
- virtual **ReturnCode_t get_uint64_value** (uint64_t ∗value, const MemberId id)=0

  Access an unsigned 64bit integer value, identified by 'id', in the sample.
- virtual **ReturnCode_t set_uint64_value** (const MemberId id, const uint64_t value)=0

  Set an unsigned 64bit integer value, identified by 'id', in the sample.
- virtual **ReturnCode_t get_float32_value** (float ∗value, const MemberId id)=0

  Access a float value, identified by 'id', in the sample.
- virtual **ReturnCode_t set_float32_value** (const MemberId id, const float value)=0

  Set a float value, identified by 'id', in the sample.
- virtual **ReturnCode_t get_float64_value** (double ∗value, const MemberId id)=0

  Access a double value, identified by 'id', in the sample.
- virtual **ReturnCode_t set_float64_value** (const MemberId id, const double value)=0

  Set a double value, identified by 'id', in the sample.
- virtual **ReturnCode_t get_char8_value** (char ∗value, const MemberId id)=0

  Access an 8bit character value, identified by 'id', in the sample.
- virtual **ReturnCode_t set_char8_value** (const MemberId id, const char value)=0

  Set an 8bit character value, identified by 'id', in the sample.
- virtual **ReturnCode_t get_char32_value** (cdx_char32_t ∗value, const MemberId id)=0

  Access a 32bit character value, identified by 'id', in the sample.
- virtual **ReturnCode_t set_char32_value** (const MemberId id, const cdx_char32_t value)=0

  Set a 32bit character value, identified by 'id', in the sample.
- virtual **ReturnCode_t get_byte_value** (unsigned char ∗value, const MemberId id)=0

  Access an 8bit value, identified by 'id', in the sample.
- virtual **ReturnCode_t set_byte_value** (const MemberId id, const unsigned char value)=0

  Set an 8bit value, identified by 'id', in the sample.
- virtual **ReturnCode_t get_boolean_value** (unsigned char ∗value, const MemberId id)=0

  Access a boolean value, identified by 'id', in the sample.
- virtual **ReturnCode_t set_boolean_value** (const MemberId id, const unsigned char value)=0

  Set a boolean value, identified by 'id', in the sample.
- virtual **ReturnCode_t get_string_value** (char ∗∗value, const MemberId id)=0

  Access a string value, identified by 'id', in the sample.
- virtual **ReturnCode_t set_string_value** (const MemberId id, const char ∗value)=0

  Set a string value, identified by 'id', in the sample.

- virtual **ReturnCode_t get_wstring_value** (cdx_char32_t **value, const MemberId id)=0

  Access a wstring value, identified by 'id', in the sample.
- virtual **ReturnCode_t set_wstring_value** (const MemberId id, const cdx_char32_t *value)=0

  Set a wstring value, identified by 'id', in the sample.
- virtual **ReturnCode_t get_complex_value** (class **DynamicData** **value, const MemberId id)=0

  Access a complex value, identified by 'id', in the sample.
- virtual **ReturnCode_t set_complex_value** (const MemberId id, class **DynamicData** *value)=0

  Set a complex value, identified by 'id', in the sample.
- virtual **ReturnCode_t get_int32_values** (Int32Seq *value, const MemberId id)=0
- virtual **ReturnCode_t set_int32_values** (const MemberId id, const Int32Seq *value)=0
- virtual **ReturnCode_t get_uint32_values** (UInt32Seq *value, const MemberId id)=0
- virtual **ReturnCode_t set_uint32_values** (const MemberId id, const UInt32Seq *value)=0
- virtual **ReturnCode_t get_int16_values** (Int16Seq *value, const MemberId id)=0
- virtual **ReturnCode_t set_int16_values** (const MemberId id, const Int16Seq *value)=0
- virtual **ReturnCode_t get_uint16_values** (UInt16Seq *value, const MemberId id)=0
- virtual **ReturnCode_t set_uint16_values** (const MemberId id, const UInt16Seq *value)=0
- virtual **ReturnCode_t get_int64_values** (Int64Seq *value, const MemberId id)=0
- virtual **ReturnCode_t set_int64_values** (const MemberId id, const Int64Seq *value)=0
- virtual **ReturnCode_t get_uint64_values** (UInt64Seq *value, const MemberId id)=0
- virtual **ReturnCode_t set_uint64_values** (const MemberId id, const UInt64Seq *value)=0
- virtual **ReturnCode_t get_float32_values** (Float32Seq *value, const MemberId id)=0
- virtual **ReturnCode_t set_float32_values** (const MemberId id, const Float32Seq *value)=0
- virtual **ReturnCode_t get_float64_values** (Float64Seq *value, const MemberId id)=0
- virtual **ReturnCode_t set_float64_values** (const MemberId id, const Float64Seq *value)=0
- virtual **ReturnCode_t get_char8_values** (CharSeq *value, const MemberId id)=0
- virtual **ReturnCode_t set_char8_values** (const MemberId id, const CharSeq *value)=0
- virtual **ReturnCode_t get_char32_values** (WcharSeq *value, const MemberId id)=0
- virtual **ReturnCode_t set_char32_values** (const MemberId id, const WcharSeq *value)=0
- virtual **ReturnCode_t get_byte_values** (ByteSeq *value, const MemberId id)=0
- virtual **ReturnCode_t set_byte_values** (const MemberId id, const ByteSeq *value)=0
- virtual **ReturnCode_t get_boolean_values** (BooleanSeq *value, const MemberId id)=0
- virtual **ReturnCode_t set_boolean_values** (const MemberId id, const BooleanSeq *value)=0
- virtual **ReturnCode_t get_string_values** (**StringSeq** *value, const MemberId id)=0
- virtual **ReturnCode_t set_string_values** (const MemberId id, const **StringSeq** *value)=0
- virtual **ReturnCode_t get_wstring_values** (WstringSeq *value, const MemberId id)=0
- virtual **ReturnCode_t set_wstring_values** (const MemberId id, const WstringSeq *value)=0

Protected Member Functions

- **DynamicData** ()

12.2.1 Detailed Description

A **DynamicData** (p. 151) object represents an individual data sample. It provides reflective getters and setters for the members of that sample.

Many of the properties and operations on **DynamicData** (p. 151) refer to values within the sample, which are identified by name, member ID, or index.

What constitutes a value within a sample, and which means of accessing it are valid, depends on the type of the sample.

- If this instance is of an aggregated type, values correspond to the type's members and can be accessed by name, member ID, or index.
- If this instance is of a sequence or string type, values correspond to the elements of the collection. These elements must be accessed by index; the mapping from index to member ID is unspecified.
- If this object is of a map type, values correspond to the values of the map. Map keys are implicitly converted to strings and can thus be used to look up map values by name. Map values can also be accessed by index, although the order is unspecified.
- If the object is of an array type, values correspond to the elements of the array. These elements must be accessed by index; the mapping from index to member ID is unspecified. If the array is multi-dimensional, elements are accessed as if they were "flattened" into a single-dimensional array in the order specified by the IDL specification.
- If the object is of a bit set type, values correspond to the flags within the bit set and are all of Boolean type. Named flags can be accessed using that name; any bit within the bound of the bit set may be accessed by its index. The mappings from name and index to member ID are unspecified.
- If the object is of an enumeration or primitive type, it has no contained values. However, the value of the sample itself may be indicated by "name" using a nil or empty string, by "ID" by passing MEMBER_ID_INVALID, or by "index" by passing index 0.

Note that indices used here are always relative to other values in a particular **DynamicData** (p. 151) object. Even though member definitions within aggregate types have a well-defined order, the same is not true within data samples or across data samples.

Specifically, the index at which a member of an aggregated type appears in a particular data sample may not match that in which it appears in the corresponding type and may not match the index at which it appears in a different data sample.

There are several reasons for these inconsistencies:

- The producer of the sample may be using a slightly different variant of the type than the consumer, which may add to, or omit elements from, the set of members known to the consumer.
- An optional member may have no value; in such a case, it will be omitted, thereby decreasing the index of every subsequent member.
- A non-optional member may likewise be omitted (which semantically is equivalent to it taking its default value). An implementation may discretionarily omit such members (e.g., to save space).
- Preserving member order is not necessary or even desirable (e.g., for performance reasons) for certain data representations.

See also

DynamicType (p. 170)
DynamicDataFactory (p. 168)

12.2.2 Constructor & Destructor Documentation

12.2.2.1 DDS::DynamicData::DynamicData ( ) [protected]

Constructor, Copy Constructor, Destructor, Assignment operator

12.2.3 Member Function Documentation

12.2.3.1 virtual ReturnCode_t DDS::DynamicData::clear_all_values ( ) [pure virtual]

Clears all values from the instance.

The meaning of "clearing" a member depends on the type of data represented by this sample:

- If this sample is of an aggregated type, and the indicated member is optional, remove it. If the indicated member is not optional, set it to its default value.
- If this sample is of a variable-length collection type, remove the indicated element, shifting any subsequent elements to the next-lowest index.
- If the sample is of an array type, set the indicated element to its default value.
- If the sample is of a bit set type, clear the indicated bit.
- If the sample is of an enumerated type, set it to the first value of the enumerated type.
- If the sample is of a primitive type, set it to its default value.
- The clear_all_members takes the above action for each value in turn.
- The clear_nonkey_value operation has exactly the same effect as clear_all_values with one exception: the values of key fields of aggregated types retain their values.

12.2.3.2 virtual ReturnCode_t DDS::DynamicData::clear_nonkey_values ( ) [pure virtual]

Clear all non-key values from the instance.

See also

DynamicData::clear_all_values (p. 155)

12.2.3.3 virtual ReturnCode_t DDS::DynamicData::clear_value (const MemberId *id*) [pure virtual]

Clear the specified value from the instance.

See also

DynamicData::clear_all_values (p. 155)

12.2.3.4 virtual class DynamicData* DDS::DynamicData::clone ( ) [pure virtual]

Create and return a new data sample with the same contents as this one.

A comparison of this object and the clone using equals immediately following this call will return true.

12.2.3.5 virtual unsigned char DDS::DynamicData::equals (const class DynamicData * *other*) [pure virtual]

Compare two **DynamicData** (p. 151) instances for equality.

Two data samples are considered to be equal if and only if all of the following conditions hold:

- Their respective type definitions are equal.
- All contained values are equal and occur in the same order.

If the samples' type is an aggregated type, the previous rule is amended as follows:

- Members shall be compared without regard to their order.
- One of the samples may omit a non-optional member that is present in the other if that member takes its default value in the latter sample.

12.2.3.6 virtual ReturnCode_t DDS::DynamicData::get_boolean_value (unsigned char * *value,* const MemberId *id*) [pure virtual]

Access a boolean value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.7 virtual ReturnCode_t DDS::DynamicData::get_boolean_values (BooleanSeq * *value,* const MemberId *id*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.8 virtual ReturnCode_t DDS::DynamicData::get_byte_value (unsigned char * *value,* const MemberId *id*) [pure virtual]

Access an 8bit value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.9 virtual ReturnCode_t DDS::DynamicData::get_byte_values (ByteSeq * *value,* const MemberId *id*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.10 virtual ReturnCode_t DDS::DynamicData::get_char32_value (cdx_char32_t * *value,* const MemberId *id*) [pure virtual]

Access a 32bit character value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.11 virtual ReturnCode_t DDS::DynamicData::get_char32_values (WcharSeq * *value,* const MemberId *id*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.12 virtual ReturnCode_t DDS::DynamicData::get_char8_value (char * *value,* const MemberId *id*) [pure virtual]

Access an 8bit character value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.13 virtual ReturnCode_t DDS::DynamicData::get_char8_values (CharSeq * *value,* const MemberId *id*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.14 virtual ReturnCode_t DDS::DynamicData::get_complex_value (class DynamicData ** *value,* const MemberId *id*) [pure virtual]

Access a complex value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.15 virtual ReturnCode_t DDS::DynamicData::get_descriptor (class MemberDescriptor * *value,* const MemberId *id*) [pure virtual]

This provides access to the descriptor for each value in this object, identified by the member ID.

See the description of **DynamicData** (p. 151) values.

12.2.3.16 virtual ReturnCode_t DDS::DynamicData::get_float32_value (float * *value,* const MemberId *id*) [pure virtual]

Access a float value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.17 virtual ReturnCode_t DDS::DynamicData::get_float32_values (Float32Seq * *value,* const MemberId *id*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.18 virtual ReturnCode_t DDS::DynamicData::get_float64_value (double * *value,* const MemberId *id*) [pure virtual]

Access a double value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.19 virtual ReturnCode_t DDS::DynamicData::get_float64_values (Float64Seq * *value,* const MemberId *id*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.20 virtual ReturnCode_t DDS::DynamicData::get_int16_value (short * *value,* const MemberId *id*) [pure virtual]

Access a 16bit integer value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.21 virtual ReturnCode_t DDS::DynamicData::get_int16_values (Int16Seq * *value,* const MemberId *id*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.22 virtual ReturnCode_t DDS::DynamicData::get_int32_value (int * *value,* const MemberId *id*) [pure virtual]

Access a 32bit integer value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.23 virtual ReturnCode_t DDS::DynamicData::get_int32_values (Int32Seq * *value,* const MemberId *id*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.24 virtual ReturnCode_t DDS::DynamicData::get_int64_value (int64_t * *value,* const MemberId *id*) [pure virtual]

Access a 64bit integer value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.25 virtual ReturnCode_t DDS::DynamicData::get_int64_values (Int64Seq * *value,* const MemberId *id*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.26 virtual unsigned int DDS::DynamicData::get_item_count ( ) [pure virtual]

Access the "item count" of the DDS_DynamicData.

The "item count" of the data depends on the type of the object.

- If the object is of a collection type, return the number of elements currently in the collection.
- In the case of an array type, this value will always be equal to the product of the bounds of all array dimensions.
- If the object is of a bit set type, return the number of named flags that are currently set in the bit set.
- If the object is of a structure or annotation type, return the number of members in the object. This value may be different than the number of members in the corresponding DDS_DynamicType; for example, some optional members may be omitted.
- If the object is of a union type, return the number of members in the object. This value will always be two: the discriminator and the current member corresponding to it.
- If the object is of a primitive or enumeration type, it is atomic, and the count is one.
- If the object is of an alias type, return the value appropriate for the alias's base type.

12.2.3.27 virtual ReturnCode_t DDS::DynamicData::get_string_value (char ** *value,* const MemberId *id*) [pure virtual]

Access a string value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.28 virtual ReturnCode_t DDS::DynamicData::get_string_values (StringSeq * *value,* const MemberId *id*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.29 virtual ReturnCode_t DDS::DynamicData::get_uint16_value (unsigned short * *value,* const MemberId *id*) [pure virtual]

Access an unsigned 16bit integer value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.30 virtual ReturnCode_t DDS::DynamicData::get_uint16_values (UInt16Seq * *value,* const MemberId *id*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.31 virtual ReturnCode_t DDS::DynamicData::get_uint32_value (unsigned int * *value,* const MemberId *id*) [pure virtual]

Access an unsigned 32bit integer value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.32 virtual ReturnCode_t DDS::DynamicData::get_uint32_values (UInt32Seq * *value,* const MemberId *id*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.33 virtual ReturnCode_t DDS::DynamicData::get_uint64_value (uint64_t * *value,* const MemberId *id*) [pure virtual]

Access an unsigned 64bit integer value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.34 virtual ReturnCode_t DDS::DynamicData::get_uint64_values (UInt64Seq * *value,* const MemberId *id*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.35 virtual ReturnCode_t DDS::DynamicData::get_wstring_value (cdx_char32_t ** *value,* const MemberId *id*) [pure virtual]

Access a wstring value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.36 virtual ReturnCode_t DDS::DynamicData::get_wstring_values (WstringSeq * *value,* const MemberId *id*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.37 virtual class DynamicData* DDS::DynamicData::loan_value (const MemberId *id*) [pure virtual]

The "loan" operations loan to the application a **DynamicData** (p. 151) object representing a value within this sample.

This operation allows applications to visit values without allocating additional **DynamicData** (p. 151) objects or copying values. This loan shall be returned by the DDS_DynamicData_return_loaned_value operation.

A given **DynamicData** (p. 151) object may support only a single outstanding loan at a time. That is, after calling a "loan" operation, an application must subsequently call return_loaned_value before calling a loan operation again. If an application violates this constraint, the loan operation shall return a nil value.

A loan operation shall also return a nil value if the indicated value does not exist.

12.2.3.38 virtual ReturnCode_t DDS::DynamicData::return_loaned_value (const class DynamicData * *value*) [pure virtual]

Return a "loaned" data sample.

The **DynamicData::return_loaned_value** (p. 162) operation may return RETCODE_PRECONDITION_NOT_MET if the provided sample object does not represent an outstanding loan from the sample on which the operation is invoked.

12.2.3.39 virtual ReturnCode_t DDS::DynamicData::set_boolean_value (const MemberId *id,* const unsigned char *value*) [pure virtual]

Set a boolean value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.40 virtual ReturnCode_t DDS::DynamicData::set_boolean_values (const MemberId *id,* const BooleanSeq * *value*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.41 virtual ReturnCode_t DDS::DynamicData::set_byte_value (const MemberId *id,* const unsigned char *value*) [pure virtual]

Set an 8bit value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.42 virtual ReturnCode_t DDS::DynamicData::set_byte_values (const MemberId *id,* const ByteSeq * *value*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.43 virtual ReturnCode_t DDS::DynamicData::set_char32_value (const MemberId *id,* const cdx_char32_t *value*) [pure virtual]

Set a 32bit character value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.44 virtual ReturnCode_t DDS::DynamicData::set_char32_values (const MemberId *id,* const WcharSeq * *value*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.45 virtual ReturnCode_t DDS::DynamicData::set_char8_value (const MemberId *id,* const char *value*) [pure virtual]

Set an 8bit character value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.46 virtual ReturnCode_t DDS::DynamicData::set_char8_values (const MemberId *id,* const CharSeq * *value*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.47 virtual ReturnCode_t DDS::DynamicData::set_complex_value (const MemberId *id,* class DynamicData * *value*) [pure virtual]

Set a complex value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.48 virtual ReturnCode_t DDS::DynamicData::set_float32_value (const MemberId *id,* const float *value*) [pure virtual]

Set a float value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.49 virtual ReturnCode_t DDS::DynamicData::set_float32_values (const MemberId *id,* const Float32Seq * *value*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.50 virtual ReturnCode_t DDS::DynamicData::set_float64_value (const MemberId *id,* const double *value*) [pure virtual]

Set a double value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.51 virtual ReturnCode_t DDS::DynamicData::set_float64_values (const MemberId *id*, const Float64Seq * *value*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.52 virtual ReturnCode_t DDS::DynamicData::set_int16_value (const MemberId *id*, const short *value*) [pure virtual]

Set a 16bit integer value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.53 virtual ReturnCode_t DDS::DynamicData::set_int16_values (const MemberId *id*, const Int16Seq * *value*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.54 virtual ReturnCode_t DDS::DynamicData::set_int32_value (const MemberId *id*, const int *value*) [pure virtual]

Set a 32bit integer value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.55 virtual ReturnCode_t DDS::DynamicData::set_int32_values (const MemberId *id*, const Int32Seq * *value*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.56 virtual ReturnCode_t DDS::DynamicData::set_int64_value (const MemberId *id,* const int64_t *value*) [pure virtual]

Set a 64bit integer value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.57 virtual ReturnCode_t DDS::DynamicData::set_int64_values (const MemberId *id,* const Int64Seq * *value*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.58 virtual ReturnCode_t DDS::DynamicData::set_string_value (const MemberId *id,* const char * *value*) [pure virtual]

Set a string value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.59 virtual ReturnCode_t DDS::DynamicData::set_string_values (const MemberId *id,* const StringSeq * *value*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.60 virtual ReturnCode_t DDS::DynamicData::set_uint16_value (const MemberId *id,* const unsigned short *value*) [pure virtual]

Set an unsigned 16bit integer value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.61 virtual ReturnCode_t DDS::DynamicData::set_uint16_values (const MemberId *id,* const UInt16Seq * *value*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.62 virtual ReturnCode_t DDS::DynamicData::set_uint32_value (const MemberId *id,* const unsigned int *value*) [pure virtual]

Set an unsigned 32bit integer value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.63 virtual ReturnCode_t DDS::DynamicData::set_uint32_values (const MemberId *id,* const UInt32Seq * *value*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.64 virtual ReturnCode_t DDS::DynamicData::set_uint64_value (const MemberId *id,* const uint64_t *value*) [pure virtual]

Set an unsigned 64bit integer value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.65 virtual ReturnCode_t DDS::DynamicData::set_uint64_values (const MemberId *id,* const UInt64Seq * *value*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.2.3.66 virtual ReturnCode_t DDS::DynamicData::set_wstring_value (const MemberId *id,* const cdx_char32_t * *value*) [pure virtual]

Set a wstring value, identified by 'id', in the sample.

If 'id' is MEMBER_ID_INVALID, then it attempts to treat 'dd' as the requested type. Otherwise, it looks in 'dd' for a value that matches that 'id'. As described above, 'id' has different meanings based on the type of 'dd'.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter or non-existent 'id'.

12.2.3.67 virtual ReturnCode_t DDS::DynamicData::set_wstring_values (const MemberId *id*, const WstringSeq * *value*) [pure virtual]

Not Yet Supported This operation is not yet implemented.

12.3 DDS::DynamicDataFactory Class Reference

This type is responsible for creating **DynamicData** (p. 151) instances.

Public Member Functions

- class **DynamicData** * **create_data** (const class **DynamicType** *type)

  *Create and return a new data sample. All objects returned by this operation should eventually be deleted by calling **DynamicDataFactory::delete_data** (p. 168).*
- **ReturnCode_t delete_data** (const class **DynamicData** *data)

  Dispose of a data sample, reclaiming any associated resources.

Static Public Member Functions

- static **DynamicDataFactory** * **get_instance** ()

  *Return a **DynamicDataFactory** (p. 168) instance that can be used to create and delete **DynamicData** (p. 151) instances.*
- static **ReturnCode_t delete_instance** ()

  Reclaim any resources associated with the object(s) previously returned from get_instance.

Protected Member Functions

- **DynamicDataFactory** ()

12.3.1 Detailed Description

This type is responsible for creating **DynamicData** (p. 151) instances.

See also

DynamicType (p. 170)
DynamicData (p. 151)

12.3.2 Constructor & Destructor Documentation

12.3.2.1 DDS::DynamicDataFactory::DynamicDataFactory ( ) [protected]

Constructor, Copy Constructor, Destructor, Assignment operator

12.3.3 Member Function Documentation

12.3.3.1 class DynamicData* DDS::DynamicDataFactory::create_data (const class DynamicType * *type*)

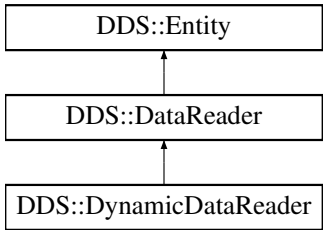
Create and return a new data sample. All objects returned by this operation should eventually be deleted by calling **DynamicDataFactory::delete_data** (p. 168).

Parameter type - The type of the sample to create.

12.4 DDS::DynamicDataReader Class Reference

Provides a **DataReader** (p. 26) interface that is tailored to support reading an X-Types defined **DynamicData** (p. 151) data type. The specific **DynamicType** (p. 170) must have been registered previously with the **DomainParticipant** (p. 44).

Inheritance diagram for DDS::DynamicDataReader:



Additional Inherited Members

12.4.1 Detailed Description

Provides a **DataReader** (p. 26) interface that is tailored to support reading an X-Types defined **DynamicData** (p. 151) data type. The specific **DynamicType** (p. 170) must have been registered previously with the **DomainParticipant** (p. 44).

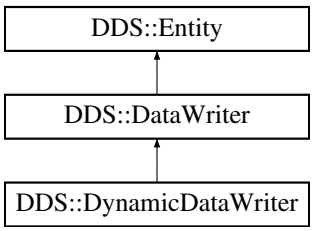
See also

DDS_DataReader

12.5 DDS::DynamicDataWriter Class Reference

A **DataWriter** (p. 37) that writes '**DynamicData** (p. 151)' data elements. Data written by this **DataWriter** (p. 37) can be read by a standard **DataReader** (p. 26) or by a **DynamicDataReader** (p. 169), if the QoS and data types match.

Inheritance diagram for DDS::DynamicDataWriter:



Additional Inherited Members

12.5.1 Detailed Description

A **DataWriter** (p. 37) that writes '**DynamicData** (p. 151)' data elements. Data written by this **DataWriter** (p. 37) can be read by a standard **DataReader** (p. 26) or by a **DynamicDataReader** (p. 169), if the QoS and data types match.

12.6 DDS::DynamicType Class Reference

An instance of **DynamicType** (p. 170) represent a type's schema: its physical name, kind, member definitions (if any), and so on.

Public Member Functions

- virtual **ReturnCode_t get_descriptor** (class **TypeDescriptor** ∗descriptor) const =0

  This operation provides a summary of the current state of this type.
- virtual const char ∗ **get_name** ()=0

  This convenience operation provides the fully qualified name of this type.
- virtual TypeKind **get_kind** ()=0

  This convenience operation returns the kind of this type (e.g., integer, structure, etc.).
- virtual **ReturnCode_t get_member_by_name** (class **DynamicTypeMember** ∗∗member, const char ∗name) const =0

  This operation accesses a member by name.
- virtual **ReturnCode_t get_all_members_by_name** (DynamicTypeMembersByName ∗member) const =0

  This operation provides access to the 'members_by_name' map.
- virtual **ReturnCode_t get_member** (class **DynamicTypeMember** ∗∗member, const MemberId id) const =0

  This operation accesses a member by id.
- virtual **ReturnCode_t get_all_members** (DynamicTypeMembersById ∗member) const =0

  This operation provides access to the 'members_by_id' map.
- virtual unsigned int **get_annotation_count** ()=0

  Return the number of annotations applied to this type.
- virtual **ReturnCode_t get_annotation** (class **AnnotationDescriptor** ∗descriptor, const unsigned int ∗idx)=0

  Access an annotation at the specified index. On success, the 'descriptor' parameter is set to the annotation.
- virtual unsigned char **equals** (const class **DynamicType** ∗other)=0

  Two types shall be considered equal if and only if all of their respective properties are equal.

Protected Member Functions

- **DynamicType** ()

12.6.1 Detailed Description

An instance of **DynamicType** (p. 170) represent a type's schema: its physical name, kind, member definitions (if any), and so on.

See also

DynamicTypeSupport (p. 188)
DynamicTypeBuilderFactory (p. 178)
DynamicTypeBuilder (p. 173)

12.6.2 Constructor & Destructor Documentation

12.6.2.1 DDS::DynamicType::DynamicType ( ) [protected]

Constructor, Copy Constructor, Destructor, Assignment operator

12.6.3 Member Function Documentation

12.6.3.1 virtual unsigned char DDS::DynamicType::equals (const class DynamicType * *other*) [pure virtual]

Two types shall be considered equal if and only if all of their respective properties are equal.

Return values

zero	if not equal
non-zero	if equal

12.6.3.2 virtual ReturnCode_t DDS::DynamicType::get_all_members (DynamicTypeMembersId * *member*) const [pure virtual]

This operation provides access to the 'members_by_id' map.

Return values

RETCODE_PRECONDITION_NOT_MET	if the type does not have members
RETCODE_BAD_PARAMETER	in the case of an invalid parameter
RETCODE_OK	on success

12.6.3.3 virtual ReturnCode_t DDS::DynamicType::get_all_members_by_name (DynamicTypeMembersByName * *member*) const [pure virtual]

This operation provides access to the 'members_by_name' map.

Return values

RETCODE_PRECONDITION_NOT_MET	if the type does not have members
RETCODE_BAD_PARAMETER	in the case of an invalid parameter
RETCODE_OK	on success

12.6.3.4 virtual ReturnCode_t DDS::DynamicType::get_annotation (class AnnotationDescriptor * *descriptor,* const unsigned int * *idx*) [pure virtual]

Access an annotation at the specified index. On success, the 'descriptor' parameter is set to the annotation.

Not Yet Supported This operation is not yet implemented.

12.6.3.5 virtual unsigned int DDS::DynamicType::get_annotation_count ( ) [pure virtual]

Return the number of annotations applied to this type.

Return values

uint	the number of annotations

Not Yet Supported This operation is not yet implemented.

12.6.3.6 virtual ReturnCode_t DDS::DynamicType::get_descriptor (class TypeDescriptor * *descriptor*) const [pure virtual]

This operation provides a summary of the current state of this type.

It populates the provided 'descriptor' parameter.

Return values

RETCODE_BAD_PARAMETER	if any parameters are invalid
RETCODE_OK	on success

12.6.3.7 virtual TypeKind DDS::DynamicType::get_kind ( ) [pure virtual]

This convenience operation returns the kind of this type (e.g., integer, structure, etc.).

Its result shall be the kind indicated by the type's descriptor property.

12.6.3.8 virtual ReturnCode_t DDS::DynamicType::get_member (class DynamicTypeMember ** *member,* const MemberId *id*) const [pure virtual]

This operation accesses a member by id.

Several types are considered to have 'members"

- structures, unions, enumerations, annotations, and bitsets

On success, the 'member' parameter is set to the member that has a matching member id.

Return values

RETCODE_PRECONDITION_NOT_MET	if the type does not have members
RETCODE_BAD_PARAMETER	in the case of an invalid parameter
RETCODE_ERROR	if the member id is not present in the type
RETCODE_OK	on success

12.6.3.9 virtual ReturnCode_t DDS::DynamicType::get_member_by_name (class DynamicTypeMember ** *member,* const char * *name*) const [pure virtual]

This operation accesses a member by name.

Several types are considered to have 'members"

- structures, unions, enumerations, annotations, and bitsets

On success, the 'member' parameter is set to the named member.

Return values

RETCODE_PRECONDITION_NOT_MET	if the type does not have members
RETCODE_BAD_PARAMETER	in the case of an invalid parameter
RETCODE_ERROR	if the named member is not present in the type
RETCODE_OK	on success

12.6.3.10 virtual const char* DDS::DynamicType::get_name ( ) [pure virtual]

This convenience operation provides the fully qualified name of this type.

It is identical to the name string that is a member of the descriptor property.

12.7 DDS::DynamicTypeBuilder Class Reference

A **DynamicTypeBuilder** (p. 173) object represents the state of a particular type defined according to the Type System. It is used to instantiate concrete **DynamicType** (p. 170) objects.

Public Member Functions

- virtual **ReturnCode_t get_descriptor** (class **TypeDescriptor** *descriptor)=0

  This operation provides a summary of the state of this type.
- virtual const char * **get_name** ()=0

  This convenience operation provides the fully qualified name of this type. I.
- virtual TypeKind **get_kind** ()=0

  This convenience operation indicates the kind of this type (e.g., integer, structure, etc.).
- virtual **ReturnCode_t get_member_by_name** (class **DynamicTypeMember** **member, const char *name)=0

  This operation accesses a member by name.

- virtual **ReturnCode_t get_all_members_by_name** (DynamicTypeMembersByName ∗member)=0

  This operation provides access to the 'members_by_name' map.
- virtual **ReturnCode_t get_member** (class **DynamicTypeMember** ∗∗member, const MemberId id)=0

  This operation accesses a member by id.
- virtual **ReturnCode_t get_all_members** (DynamicTypeMembersById ∗member)=0

  This operation provides access to the 'members_by_id' map.
- virtual unsigned int **get_annotation_count** ()=0

  Return the number of annotations applied to this type.
- virtual **ReturnCode_t get_annotation** (class **AnnotationDescriptor** ∗descriptor, const unsigned int idx)=0

  Access an annotation at the specified index. On success, the 'descriptor' parameter is set to the annotation.
- virtual unsigned char **equals** (const class **DynamicType** ∗other)=0

  Two types shall be considered equal if and only if all of their respective properties are equal.
- virtual **ReturnCode_t add_member** (const class **MemberDescriptor** ∗descriptor)=0

  Add a member to the type.
- virtual **ReturnCode_t apply_annotation** (const class **AnnotationDescriptor** ∗descriptor)=0

  Apply the given annotation to this type.
- virtual class **DynamicType** ∗ **build** ()=0

  *Create an immutable **DynamicType** (p. 170) object defined by the builder's current state.*
- virtual **ReturnCode_t delete_type** (class **DynamicType** ∗type)=0

  *Destroy the type information contained in this **DynamicTypeBuilder** (p. 173) instance.*

Protected Member Functions

- **DynamicTypeBuilder** ()

12.7.1 Detailed Description

A **DynamicTypeBuilder** (p. 173) object represents the state of a particular type defined according to the Type System. It is used to instantiate concrete **DynamicType** (p. 170) objects.

See also

DynamicType (p. 170)
DynamicTypeBuilderFactory (p. 178)
DynamicTypeSupport (p. 188)

12.7.2 Constructor & Destructor Documentation

12.7.2.1 DDS::DynamicTypeBuilder::DynamicTypeBuilder ( ) [protected]

Constructor, Copy Constructor, Destructor, Assignment operator

12.7.3 Member Function Documentation

12.7.3.1 virtual ReturnCode_t DDS::DynamicTypeBuilder::add_member (const class MemberDescriptor ∗ *descriptor*) [pure virtual]

Add a member to the type.

Return values

RETCODE_PRECONDITION_NOT_MET	if the type does not have members or a member with the same id already exists
RETCODE_BAD_PARAMETER	in the case of an invalid parameter
RETCODE_OUT_OF_RESOURCES	in the case of a memory shortage
RETCODE_OK	on success

12.7.3.2 virtual ReturnCode_t DDS::DynamicTypeBuilder::apply_annotation (const class AnnotationDescriptor * *descriptor*) [pure virtual]

Apply the given annotation to this type.

Parameter descriptor - A consistent descriptor for the annotation to apply.

Return values

RETCODE_BAD_PARAMETER	If this 'descriptor' parameter is not consistent

Not Yet Supported This operation is not yet implemented.

12.7.3.3 virtual class DynamicType* DDS::DynamicTypeBuilder::build ( ) [pure virtual]

Create an immutable **DynamicType** (p. 170) object defined by the builder's current state.

Subsequent changes to this builder, if any, shall have no effect on the state of any previously created **DynamicType** (p. 170).

Return values

nil	in the case of an error

12.7.3.4 virtual ReturnCode_t DDS::DynamicTypeBuilder::delete_type (class DynamicType * *type*) [pure virtual]

Destroy the type information contained in this **DynamicTypeBuilder** (p. 173) instance.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter
RETCODE_OK	on success

12.7.3.5 virtual unsigned char DDS::DynamicTypeBuilder::equals (const class DynamicType * *other*) [pure virtual]

Two types shall be considered equal if and only if all of their respective properties are equal.

Return values

zero	if not equal
non-zero	if equal

12.7.3.6 virtual ReturnCode_t DDS::DynamicTypeBuilder::get_all_members (DynamicTypeMembersId * *member*) [pure virtual]

This operation provides access to the 'members_by_id' map.

Return values

RETCODE_PRECONDITION_NOT_MET	if the type does not have members
RETCODE_BAD_PARAMETER	in the case of an invalid parameter
DDRETCODE_OK	on success

12.7.3.7 virtual ReturnCode_t DDS::DynamicTypeBuilder::get_all_members_by_name (DynamicTypeMembersByName * *member*) [pure virtual]

This operation provides access to the 'members_by_name' map.

Return values

RETCODE_PRECONDITION_NOT_MET	if the type does not have members
RETCODE_BAD_PARAMETER	in the case of an invalid parameter
RETCODE_OK	on success

12.7.3.8 virtual ReturnCode_t DDS::DynamicTypeBuilder::get_annotation (class AnnotationDescriptor * *descriptor,* const unsigned int *idx*) [pure virtual]

Access an annotation at the specified index. On success, the 'descriptor' parameter is set to the annotation.

Not Yet Supported This operation is not yet implemented.

12.7.3.9 virtual unsigned int DDS::DynamicTypeBuilder::get_annotation_count ( ) [pure virtual]

Return the number of annotations applied to this type.

Return values

uint	the number of annotations

Not Yet Supported This operation is not yet implemented.

12.7.3.10 virtual ReturnCode_t DDS::DynamicTypeBuilder::get_descriptor (class TypeDescriptor * *descriptor*) [pure virtual]

This operation provides a summary of the state of this type.

It overwrites the state of the application-provided 'descriptor' object.

Return values

RETCODE_BAD_PARAMETER	if the descriptor parameter is invalid
RETCODE_OK	on success

12.7.3.11 virtual TypeKind DDS::DynamicTypeBuilder::get_kind ( ) [pure virtual]

This convenience operation indicates the kind of this type (e.g., integer, structure, etc.).

Its result shall be the same as the kind indicated by the type's descriptor property.

12.7.3.12 virtual ReturnCode_t DDS::DynamicTypeBuilder::get_member (class DynamicTypeMember ** *member,* const MemberId *id*) [pure virtual]

This operation accesses a member by id.

Several types are considered to have 'members"

- structures, unions, enumerations, annotations, and bitsets

On success, the 'member' parameter is set to the member that has a matching member id.

Return values

RETCODE_PRECONDITION_NOT_MET	if the type does not have members
RETCODE_BAD_PARAMETER	in the case of an invalid parameter
RETCODE_ERROR	if the member id is not present in the type
RETCODE_OK	on success

12.7.3.13 virtual ReturnCode_t DDS::DynamicTypeBuilder::get_member_by_name (class DynamicTypeMember ** *member,* const char * *name*) [pure virtual]

This operation accesses a member by name.

Several types are considered to have 'members"

- structures, unions, enumerations, annotations, and bitsets

On success, the 'member' parameter is set to the named member.

Return values

RETCODE_PRECONDITION_NOT_MET	if the type does not have members

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter
RETCODE_ERROR	if the named member is not present in the type
RETCODE_OK	on success

12.7.3.14 virtual const char∗ DDS::DynamicTypeBuilder::get_name ( ) [pure virtual]

This convenience operation provides the fully qualified name of this type. I.

t shall be identical to the name string that is a member of the descriptor property.

12.8 DDS::DynamicTypeBuilderFactory Class Reference

An instance of this type is responsible for creating **DynamicType** (p. 170) and **DynamicTypeSupport** (p. 188) objects.

Public Member Functions

- virtual class **DynamicType** ∗ **get_primitive_type** (const TypeKind kind)=0

  *Retrieve a **DynamicType** (p. 170) object corresponding to the indicated primitive type kind.*
- virtual class **DynamicTypeBuilder** ∗ **create_type** (const class **TypeDescriptor** ∗descriptor)=0

  Create and return a new DDS_DynamicTypeBuilder object as described by the given type descriptor.
- virtual class **DynamicTypeBuilder** ∗ **create_type_copy** (const class **DynamicType** ∗type)=0

  Create and return a new DDS_DynamicTypeBuilder object with a copy of the state of the given type.
- virtual class **DynamicTypeBuilder** ∗ **create_type_w_type_object** (const struct TypeObject ∗type_object)=0

  *Create and return a new **DynamicTypeBuilder** (p. 173) object that describes a type identical to that described by the given TypeObject.*
- virtual class **DynamicTypeBuilder** ∗ **create_string_type** (const unsigned int bound)=0

  *Create and return a new **DynamicTypeBuilder** (p. 173) object representing a string type.*
- virtual class **DynamicTypeBuilder** ∗ **create_wstring_type** (const unsigned int bound)=0

  *Create and return a new **DynamicTypeBuilder** (p. 173) object representing a string type.*
- virtual class **DynamicTypeBuilder** ∗ **create_sequence_type** (const class **DynamicType** ∗element_type, const unsigned int bound)=0

  *Create and return a new **DynamicTypeBuilder** (p. 173) object representing a sequence type.*
- virtual class **DynamicTypeBuilder** ∗ **create_array_type** (const class **DynamicType** ∗element_type, const BoundSeq ∗bound)=0

  *Create and return a new **DynamicTypeBuilder** (p. 173) object representing an array type.*
- virtual class **DynamicTypeBuilder** ∗ **create_map_type** (const class **DynamicType** ∗key_element_type, const class **DynamicType** ∗element_type, const unsigned int bound)=0

  *Create and return a new **DynamicTypeBuilder** (p. 173) object representing a map type.*
- virtual class **DynamicTypeBuilder** ∗ **create_bitset_type** (const unsigned int bound)=0

  *Create and return a new **DynamicTypeBuilder** (p. 173) object representing a bit set type.*
- virtual class **DynamicTypeBuilder** ∗ **create_type_w_uri** (const char ∗document_url, const char ∗type_name, const IncludePathSeq ∗include_paths)=0

  *Create and return a new **DynamicTypeBuilder** (p. 173) object by parsing the type description at the given URL.*
- virtual class **DynamicTypeBuilder** ∗ **create_type_w_document** (const char ∗document, const char ∗type_name, const IncludePathSeq ∗include_paths)=0

*Create and return a new **DynamicTypeBuilder** (p. 173) object by parsing the type description contained in the given string.*

- virtual class **DynamicTypeBuilder** ∗ **create_structure_type** ()=0

  *Create and return a new **DynamicTypeBuilder** (p. 173) object representing a structure type.*

- virtual class **DynamicTypeBuilder** ∗ **create_union_type** ()=0

  *Create and return a new **DynamicTypeBuilder** (p. 173) object representing a union type.*

- virtual class **DynamicTypeBuilder** ∗ **create_alias_type** (const **DynamicType** ∗base_type)=0

  Create and return a new DDS_DynamicTypeBuilder object representing an alias for the provided 'base_type'.

- virtual class **DynamicTypeBuilder** ∗ **create_enumeration_type** (const unsigned int bound=32)=0

  *Create and return a new **DynamicTypeBuilder** (p. 173) object representing an enumeration type.*

- virtual **ReturnCode_t delete_type_builder** (class **DynamicTypeBuilder** ∗dtb)=0

  *Destroy a **DynamicTypeBuilder** (p. 173) instance obtained through one of the create methods on the **DynamicTypeBuilderFactory** (p. 178).*

- virtual **ReturnCode_t delete_type** (class **DynamicType** ∗type)=0

  *Destroy a **DynamicType** (p. 170) instance obtained through the **DynamicTypeBuilderFactory::get_primitive_type** (p. 185) operation.*

Static Public Member Functions

- static **DynamicTypeBuilderFactory** ∗ **get_instance** ()

  Return the DDS_DynamicTypeBuilderFactory singleton instance.

- static **ReturnCode_t delete_instance** ()

  *Reclaim any resources associated with the instance returned from **DynamicTypeBuilderFactory::get_instance** (p. 185).*

Protected Member Functions

- **DynamicTypeBuilderFactory** ()

12.8.1 Detailed Description

An instance of this type is responsible for creating **DynamicType** (p. 170) and **DynamicTypeSupport** (p. 188) objects.

See also

DynamicType (p. 170)
DynamicTypeBuilder (p. 173)
DynamicTypeSupport (p. 188)

12.8.2 Constructor & Destructor Documentation

12.8.2.1 DDS::DynamicTypeBuilderFactory::DynamicTypeBuilderFactory ( ) [protected]

Constructor, Copy Constructor, Destructor, Assignment operator

12.8.3 Member Function Documentation

12.8.3.1 virtual class DynamicTypeBuilder* DDS::DynamicTypeBuilderFactory::create_alias_type (const DynamicType * *base_type*) [pure virtual]

Create and return a new DDS_DynamicTypeBuilder object representing an alias for the provided 'base_type'.

This is a convenience routine, in place of calling DynamicTypeBuilder::create_type() directly. The alias type refers to the 'base_type' parameter.

Return values

nil	In the case of an error

12.8.3.2 virtual class DynamicTypeBuilder* DDS::DynamicTypeBuilderFactory::create_array_type (const class DynamicType * *element_type,* const BoundSeq * *bound*) [pure virtual]

Create and return a new **DynamicTypeBuilder** (p. 173) object representing an array type.

All objects returned by this operation should eventually be deleted by calling **DynamicTypeBuilderFactory::delete_type_builder** (p. 185).

All array types having equal element types, an equal number of dimensions, and equal bounds in each dimension shall be considered equal.

The type of all objects that can be stored in an array of the new type is specified by parameter 'element_type'.

The 'bound' parameter is a collection of unsigned integers, the length of which is equal to the number of dimensions in the new array type, and the values of which are the bounds of each dimension. (For example, a three-by-two array would be described by a collection of length two, where the first element had a value of three and the second a value of two.)

Return values

nil	If any parameters are invalid or an error occurs.

12.8.3.3 virtual class DynamicTypeBuilder* DDS::DynamicTypeBuilderFactory::create_bitset_type (const unsigned int *bound*) [pure virtual]

Create and return a new **DynamicTypeBuilder** (p. 173) object representing a bit set type.

All objects returned by this operation should eventually be deleted by calling **DynamicTypeBuilderFactory::delete_type_builder** (p. 185).

Parameter bound - The number of reserved bits in the bit set.

Return values

nil	If an error occurs, for example if the bound is out of range.

12.8.3.4 virtual class DynamicTypeBuilder* DDS::DynamicTypeBuilderFactory::create_enumeration_type (const unsigned int *bound* = 32) [pure virtual]

Create and return a new **DynamicTypeBuilder** (p. 173) object representing an enumeration type.

This is a convenience routine, in place of calling DynamicTypeBuilder::create_type() directly. The enumeration type of the result returned shall initially contain no members.

Return values

nil	In the case of an error

12.8.3.5 virtual class DynamicTypeBuilder* DDS::DynamicTypeBuilderFactory::create_map_type (const class DynamicType * *key_element_type,* const class DynamicType * *element_type,* const unsigned int *bound*) [pure virtual]

Create and return a new **DynamicTypeBuilder** (p. 173) object representing a map type.

All objects returned by this operation should eventually be deleted by calling **DynamicTypeBuilderFactory::delete_type_builder** (p. 185).

All map types having equal key and value element types and equal bounds shall be considered equal.

Parameter key_element_type - The type of all objects that can be stored as keys in a map of the new type.

Parameter element_type - The type of all objects that can be stored as values in a map of the new type.

Parameter bound - The maximum number of key-value pairs that may be stored in a map of the new type. If this argument is equal to LENGTH_UNLIMITED, the map type shall be considered to be unbounded.

Return values

nil	If an error occurs, or any parameter is invalid

12.8.3.6 virtual class DynamicTypeBuilder* DDS::DynamicTypeBuilderFactory::create_sequence_type (const class DynamicType * *element_type,* const unsigned int *bound*) [pure virtual]

Create and return a new **DynamicTypeBuilder** (p. 173) object representing a sequence type.

All objects returned by this operation should eventually be deleted by calling **DynamicTypeBuilderFactory::delete_type_builder** (p. 185).

All sequence types having equal element types and equal bounds shall be considered equal.

The type of all objects that can be stored in a sequence of the new type is specified by the 'element_type' parameter.

The maximum number of elements that may be stored in a sequence of the new type is specified by the 'bound' parameter. If this argument is equal to DDS_LENGTH_UNLIMITED, the sequence type shall be considered to be unbounded.

Return values

nil	If an error occurs, or any parameters are invalid

12.8.3.7 virtual class DynamicTypeBuilder* DDS::DynamicTypeBuilderFactory::create_string_type (const unsigned int *bound*) [pure virtual]

Create and return a new **DynamicTypeBuilder** (p. 173) object representing a string type.

The element type of the result returned by this operation is Char8.

All string types having equal element types and equal bounds shall be considered equal.

The maximum number of elements that may be stored in a string of the new type is specified by parameter 'bound'. If this argument is equal to LENGTH_UNLIMITED, the string type shall be unbounded.

Return values

nil	If an error occurs

12.8.3.8 virtual class DynamicTypeBuilder* DDS::DynamicTypeBuilderFactory::create_structure_type ( ) [pure virtual]

Create and return a new **DynamicTypeBuilder** (p. 173) object representing a structure type.

This is a convenience routine, in place of calling DynamicTypeBuilder::create_type() directly. The structure type of the result returned shall initially contain no members.

Return values

nil	In the case of an error a

12.8.3.9 virtual class DynamicTypeBuilder* DDS::DynamicTypeBuilderFactory::create_type (const class TypeDescriptor * *descriptor*) [pure virtual]

Create and return a new DDS_DynamicTypeBuilder object as described by the given type descriptor.

This method is the conventional mechanism for creating structured, enumeration, and alias types, although it can also be used to create types of other kinds. All objects returned by this operation should eventually be deleted by calling **DynamicTypeBuilderFactory::delete_type** (p. 185).

The properties of the new type to create is provided by the 'descriptor' parameter.

Return values

nil	If the descriptor parameter is nil or inconsistent (as indicated by its is_consistent operation)

12.8.3.10 virtual class DynamicTypeBuilder* DDS::DynamicTypeBuilderFactory::create_type_copy (const class DynamicType * *type*) [pure virtual]

Create and return a new DDS_DynamicTypeBuilder object with a copy of the state of the given type.

All objects returned by this operation should eventually be deleted by calling **DynamicTypeBuilderFactory::delete_type_builder** (p. 185). The initial state of the new type to create is described by the parameter 'type'.

Return values

nil	If parameter 'type' is nil

12.8.3.11 virtual class DynamicTypeBuilder* DDS::DynamicTypeBuilderFactory::create_type_w_document (const char * *document,* const char * *type_name,* const IncludePathSeq * *include_paths*) [pure virtual]

Create and return a new **DynamicTypeBuilder** (p. 173) object by parsing the type description contained in the given string.

Applications shall be able to reclaim resources associated with the type returned by this method by calling **DynamicTypeBuilderFactory::delete_type_builder** (p. 185), just as if the resultant type was created by one of the create methods of this class.

Parameter document - A type description document, which shall be parsed to create the **DynamicType** (p. 170) object. Implementations shall minimally support the XML Type Description format for loaded documents and may support additional Type Descriptions.

Parameter type_name - The fully qualified name of the type to be loaded from the document. If no type exists of this name in the document, the operation shall fail and return a nil result.

Parameter include_paths - A collection of URLs to directories to be searched for additional type description documents that may be included, directly or indirectly, by the document argument. Implementations shall minimally support the file: URL scheme and may support additional schemes.

Return values

nil	If an error occurs

Not Yet Supported This operation is not yet implemented.

12.8.3.12 virtual class DynamicTypeBuilder* DDS::DynamicTypeBuilderFactory::create_type_w_type_object (const struct TypeObject * *type_object*) [pure virtual]

Create and return a new **DynamicTypeBuilder** (p. 173) object that describes a type identical to that described by the given TypeObject.

Subsequent changes to the new DDS_DynamicTypeBuilder object shall not impact the input TypeObject 'type_object'. All objects returned by this operation should eventually be deleted by calling **DynamicTypeBuilderFactory::delete_type_builder** (p. 185).

Return values

nil	If the type_object parameter is invalid

12.8.3.13 virtual class DynamicTypeBuilder* DDS::DynamicTypeBuilderFactory::create_type_w_uri (const char * *document_url,* const char * *type_name,* const IncludePathSeq * *include_paths*) [pure virtual]

Create and return a new **DynamicTypeBuilder** (p. 173) object by parsing the type description at the given URL.

Applications shall be able to reclaim resources associated with the type returned by this method by calling **DynamicTypeBuilderFactory::delete_type_builder** (p. 185), just as if the resultant type was created by one of the create methods of this class.

Parameter document_url - A URL that indicates a type description document, which shall be parsed to create the **DynamicType** (p. 170) object. Implementations shall minimally support the file: URL scheme and may support additional schemes. Implementations shall minimally support the XML Type Description format for loaded documents and may support additional Type Descriptions.

Parameter type_name - The fully qualified name of the type to be loaded from the document that is the target of the URL. If no type exists of this name in the document, the operation shall fail and return a nil result.

Parameter include_paths - A collection of URLs to directories to be searched for additional type description documents that may be included, directly or indirectly, by the document that is the target of document_url. The directory in which the target of document_url resides shall be considered on the inclusion search path implicitly and need not be included in this collection. Implementations shall minimally support the file: URL scheme and may support additional schemes.

Return values

nil	If an error occurs, or if any parameters are invalid

Not Yet Supported This operation is not yet implemented.

12.8.3.14 virtual class DynamicTypeBuilder* DDS::DynamicTypeBuilderFactory::create_union_type ( ) [pure virtual]

Create and return a new **DynamicTypeBuilder** (p. 173) object representing a union type.

This is a convenience routine, in place of calling DynamicTypeBuilder::create_type() directly. The union type of the result returned shall initially contain no members.

Return values

nil	In the case of an error

12.8.3.15 virtual class DynamicTypeBuilder* DDS::DynamicTypeBuilderFactory::create_wstring_type (const unsigned int *bound*) [pure virtual]

Create and return a new **DynamicTypeBuilder** (p. 173) object representing a string type.

The element type of the result returned by this operation is Char32.

All string types having equal element types and equal bounds shall be considered equal.

The maximum number of elements that may be stored in a string of the new type is specified by parameter 'bound'. If this argument is equal to LENGTH_UNLIMITED, the string type shall be unbounded.

Return values

nil	If an error occurs

12.8.3.16 static ReturnCode_t DDS::DynamicTypeBuilderFactory::delete_instance ( ) [static]

Reclaim any resources associated with the instance returned from **DynamicTypeBuilderFactory::get_instance** (p. 185).

Return values

RETCODE_OK	on success
RETCODE_ERROR	if it fails

12.8.3.17 virtual ReturnCode_t DDS::DynamicTypeBuilderFactory::delete_type (class DynamicType * *type*) [pure virtual]

Destroy a **DynamicType** (p. 170) instance obtained through the **DynamicTypeBuilderFactory::get_primitive_type** (p. 185) operation.

Return values

RETCODE_OK	on success
RETCODE_ALREADY_DELETED	if the type has previously been deleted and the middlware can detect that fact
RETCODE_ERROR	on any other error

12.8.3.18 virtual ReturnCode_t DDS::DynamicTypeBuilderFactory::delete_type_builder (class DynamicTypeBuilder * *dtb*) [pure virtual]

Destroy a **DynamicTypeBuilder** (p. 173) instance obtained through one of the create methods on the **DynamicTypeBuilderFactory** (p. 178).

Return values

RETCODE_OK	on success
RETCODE_ERROR	on any other error

12.8.3.19 static DynamicTypeBuilderFactory* DDS::DynamicTypeBuilderFactory::get_instance ( ) [static]

Return the DDS_DynamicTypeBuilderFactory singleton instance.

Calling this operation is legal even after **DynamicTypeBuilderFactory::delete_instance** (p. 185) has been called.

In such a case, the implementation will recreate or restore the state of the "singleton" as necessary in order to return a valid object to the caller.

If an error occurs, this method shall return a nil value.

12.8.3.20 virtual class DynamicType* DDS::DynamicTypeBuilderFactory::get_primitive_type (const TypeKind *kind*) [pure virtual]

Retrieve a **DynamicType** (p. 170) object corresponding to the indicated primitive type kind.

The kind of the primitive type whose representation is to be returned is specified by parameter 'kind'.

Return values

nil	if the given kind does not correspond to a primitive type

12.9 DDS::DynamicTypeMember Class Reference

A **DynamicTypeMember** (p. 186) represents a "member" of a type. A "member" in this sense may be a member of an aggregated type, a constant within an enumeration, or some other type substructure.

Public Member Functions

- virtual **ReturnCode_t get_descriptor** (class **MemberDescriptor** ∗descriptor)=0

  This operation accesses the current state of this type.
- virtual unsigned int **get_annotation_count** ()=0

  Return the number of annotations applied to this type.
- virtual **ReturnCode_t get_annotation** (class **AnnotationDescriptor** ∗descriptor, const unsigned int idx)=0

  Access an annotation at the specified index. On success, the 'descriptor' parameter is set to the annotation.
- virtual unsigned char **equals** (const class **DynamicTypeMember** ∗other)=0

  *Compare two **DynamicTypeMember** (p. 186)'s.*
- virtual MemberId **get_id** ()=0

  This convenience operation provides the member ID of this member.
- virtual const char ∗ **get_name** ()=0

  This convenience operation provides the name of this member.
- virtual MemberFlag **get_flags** ()=0

  Access the type flags.
- virtual **ReturnCode_t set_flags** (MemberFlag f)=0

  Set the type flags.

Protected Member Functions

- **DynamicTypeMember** ()

12.9.1 Detailed Description

A **DynamicTypeMember** (p. 186) represents a "member" of a type. A "member" in this sense may be a member of an aggregated type, a constant within an enumeration, or some other type substructure.

See also

DynamicType (p. 170)
DynamicTypeBuilderFactory (p. 178)
DynamicTypeBuilder (p. 173)

12.9.2 Constructor & Destructor Documentation

12.9.2.1 DDS::DynamicTypeMember::DynamicTypeMember ( ) [protected]

Constructor, Copy Constructor, Destructor, Assignment operator

12.9.3 Member Function Documentation

12.9.3.1 virtual unsigned char DDS::DynamicTypeMember::equals (const class DynamicTypeMember * *other*) [pure virtual]

Compare two **DynamicTypeMember** (p. 186)'s.

Two type descriptors are considered equal if and only if if they belong to the same type and the values of all of the properties are equal in each of them.

12.9.3.2 virtual ReturnCode_t DDS::DynamicTypeMember::get_annotation (class AnnotationDescriptor * *descriptor,* const unsigned int *idx*) [pure virtual]

Access an annotation at the specified index. On success, the 'descriptor' parameter is set to the annotation.

Not Yet Supported This operation is not yet implemented.

12.9.3.3 virtual unsigned int DDS::DynamicTypeMember::get_annotation_count ( ) [pure virtual]

Return the number of annotations applied to this type.

Return values

uint	the number of annotations

Not Yet Supported This operation is not yet implemented.

12.9.3.4 virtual ReturnCode_t DDS::DynamicTypeMember::get_descriptor (class MemberDescriptor * *descriptor*) [pure virtual]

This operation accesses the current state of this type.

It overwrites the application-provided 'descriptor' parameter.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter

12.9.3.5 virtual MemberId DDS::DynamicTypeMember::get_id ( ) [pure virtual]

This convenience operation provides the member ID of this member.

Its result shall be identical to the ID value that is a member of the descriptor property.

12.9.3.6 virtual const char∗ DDS::DynamicTypeMember::get_name ( ) [pure virtual]

This convenience operation provides the name of this member.

Its result shall be identical to the name string that is a member of the descriptor property.

12.10 DDS::DynamicTypeSupport Class Reference

The **DynamicTypeSupport** (p. 188) interface extends the TypeSupport interface defined by the **DDS** (p. 313) specification. This TypeSupport operates on **DynamicData** (p. 151) samples.

Inherits TypeSupport.

Public Member Functions

- virtual **ReturnCode_t register_type** (**DomainParticipant** ∗participant, const char ∗type_name)=0

  *Register the TypeSupport with the **DomainParticipant** (p. 44).*
- virtual const char ∗ **get_type_name** ()=0

  Returns the name of the type supported by this TypeSupport instance.
- virtual **DynamicType** ∗ **get_type** ()=0

  *Returns the **DynamicType** (p. 170) that is supported by this TypeSupport instance.*

Static Public Member Functions

- static **DynamicTypeSupport** ∗ **create_type_support** (**DynamicType** ∗type)

  *Create and return a new **DynamicTypeSupport** (p. 188) object capable of supporting the given type.*
- static **ReturnCode_t delete_type_support** (**DynamicTypeSupport** ∗type_support)

  *Delete the given type support object, which was previously created by a call to **DynamicTypeSupport::create_type_support** (p. 189).*

Protected Member Functions

- **DynamicTypeSupport** ()

12.10.1 Detailed Description

The **DynamicTypeSupport** (p. 188) interface extends the TypeSupport interface defined by the **DDS** (p. 313) specification. This TypeSupport operates on **DynamicData** (p. 151) samples.

See also

DynamicType (p. 170)
DynamicTypeBuilderFactory (p. 178)
DynamicTypeBuilder (p. 173)

12.10.2 Constructor & Destructor Documentation

12.10.2.1 DDS::DynamicTypeSupport::DynamicTypeSupport ( ) [protected]

Constructor, Copy Constructor, Destructor, Assignment operator

12.10.3 Member Function Documentation

12.10.3.1 static DynamicTypeSupport* DDS::DynamicTypeSupport::create_type_support (DynamicType * *type*) [static]

Create and return a new **DynamicTypeSupport** (p. 188) object capable of supporting the given type.

The new type support has a "copy" of the given 'type' parameter, such that subsequent changes to, or deletions of, the 'type' parameter do not impact the new type support.

Any object returned by this operation should eventually be deleted by calling **DynamicTypeSupport::delete_type_support** (p. 189).

Parameter type - The type for which to create a type support.

Return values

nil	If an error occurs, or the parameter is invalid

12.10.3.2 static ReturnCode_t DDS::DynamicTypeSupport::delete_type_support (DynamicTypeSupport * *type_support*) [static]

Delete the given type support object, which was previously created by a call to **DynamicTypeSupport::create_type_support** (p. 189).

Return values

DDS_RETCODE_BAD_PARAMETER	invalid parameter
DDS_RETCODE_OK	on success

12.10.3.3 virtual DynamicType* DDS::DynamicTypeSupport::get_type ( ) [pure virtual]

Returns the **DynamicType** (p. 170) that is supported by this TypeSupport instance.

The application should not modify the returned **DynamicType** (p. 170).

12.10.3.4 virtual const char* DDS::DynamicTypeSupport::get_type_name ( ) [pure virtual]

Returns the name of the type supported by this TypeSupport instance.

The returned string is owned by the TypeSupport, and must not be free'd or modified.

12.11 DDS::MemberDescriptor Class Reference

A **MemberDescriptor** (p. 190) comprises the state of a **DynamicTypeMember** (p. 186).

Public Member Functions

- **MemberDescriptor** ()
- **ReturnCode_t copy_from** (const class **MemberDescriptor** ∗descriptor)

  Overwrite the contents of the 'to' descriptor with those of the provided 'from' descriptor.
- unsigned char **equals** (const class **MemberDescriptor** ∗descriptor)

  *Compare two **TypeDescriptor** (p. 191)'s.*
- unsigned char **is_consistent** ()

  Indicates whether the states of all of this descriptor's properties are consistent.

12.11.1 Detailed Description

A **MemberDescriptor** (p. 190) comprises the state of a **DynamicTypeMember** (p. 186).

12.11.2 Constructor & Destructor Documentation

12.11.2.1 DDS::MemberDescriptor::MemberDescriptor ( )

Constructor, Copy Constructor, Destructor, Assignment operator

12.11.3 Member Function Documentation

12.11.3.1 ReturnCode_t DDS::MemberDescriptor::copy_from (const class MemberDescriptor ∗ *descriptor*)

Overwrite the contents of the 'to' descriptor with those of the provided 'from' descriptor.

Subsequent calls to equals, passing the same argument as to this method, return true.

The 'from' descriptor shall not be changed by this operation.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter
RETCODE_ERROR	in the case of any other error

12.11.3.2 unsigned char DDS::MemberDescriptor::equals (const class MemberDescriptor ∗ *descriptor*)

Compare two **TypeDescriptor** (p. 191)'s.

Two type descriptors are considered equal if and only if the values of all of the properties identified in the table above are equal in each of them.

12.11.3.3 unsigned char DDS::MemberDescriptor::is_consistent ( )

Indicates whether the states of all of this descriptor's properties are consistent.

The definitions of consistency for each property are given in the section corresponding to that property.

Return values

non-zero	if descriptor is consistent
zero	if descriptor is not consistent

12.12 DDS::TypeDescriptor Class Reference

A **TypeDescriptor** (p. 191) comprises the state of a type.

Public Member Functions

- **TypeDescriptor** ()
- **ReturnCode_t copy_from** (const class **TypeDescriptor** ∗other)

  Overwrite the contents of this descriptor with those of another descriptor.
- unsigned char **equals** (const class **TypeDescriptor** ∗other)

  *Compare two **TypeDescriptor** (p. 191)'s.*
- unsigned char **is_consistent** ()

  Indicates whether the states of all of this descriptor's properties are consistent.

Public Attributes

- TypeKind **kind**

  the 'kind' of this type
- ObjectName **name**

  the 'name' of this type
- class **DynamicType** ∗ **base_type**

  the 'base_type' of this type
- class **DynamicType** ∗ **discriminator_type**

  If this descriptor represents a union type, this field indicates the type of the discriminator of the union.
- BoundSeq **bound**

  The bound property indicates the bound of collection and similar types.
- class **DynamicType** ∗ **element_type**

  If this descriptor represents an array, sequence, or string type, this property indicates the element type of the collection.
- class **DynamicType** ∗ **key_element_type**

  If this descriptor represents a map type, this property indicates the value element type of the map.

12.12.1 Detailed Description

A **TypeDescriptor** (p. 191) comprises the state of a type.

12.12.2 Constructor & Destructor Documentation

12.12.2.1 DDS::TypeDescriptor::TypeDescriptor ( )

Constructor, Copy Constructor, Destructor, Assignment operator

12.12.3 Member Function Documentation

12.12.3.1 ReturnCode_t DDS::TypeDescriptor::copy_from (const class TypeDescriptor * *other*)

Overwrite the contents of this descriptor with those of another descriptor.

Subsequent calls to equals, passing the same argument as to this method, return true. The other descriptor shall not be changed by this operation.

Return values

RETCODE_BAD_PARAMETER	in the case of an invalid parameter
RETCODE_ERROR	in the case of any other error

12.12.3.2 unsigned char DDS::TypeDescriptor::equals (const class TypeDescriptor * *other*)

Compare two **TypeDescriptor** (p. 191)'s.

Two type descriptors are considered equal if and only if the values of all of the properties identified in the table above are equal in each of them.

12.12.3.3 unsigned char DDS::TypeDescriptor::is_consistent ( )

Indicates whether the states of all of this descriptor's properties are consistent.

The definitions of consistency for each property are given in the section corresponding to that property.

Return values

non-zero	if descriptor is consistent
zero	if descriptor is not consistent

12.12.4 Member Data Documentation

12.12.4.1 class DynamicType* DDS::TypeDescriptor::base_type

the 'base_type' of this type

- If this descriptor represents a structure type, base_type indicates the supertype of that type. A nil value of this property indicates that the structure type has no supertype.
- If this descriptor represents an alias type, base_type indicates the type being aliased. A nil value for this property is not considered consistent.

In all other cases, a consistent descriptor shall have a nil value for this property.

12.12.4.2 BoundSeq DDS::TypeDescriptor::bound

The bound property indicates the bound of collection and similar types.

- If this descriptor represents an array type, the length of the property value indicates the number of dimensions in the array, and each value indicates the bound of the corresponding dimension.
- If this descriptor represents a sequence, map, bit set, or string type, the length of the property value is one and the integral value in that property indicates the bound of the collection.

In all other cases, a consistent descriptor shall have a nil value for this property.

12.12.4.3 class DynamicType* DDS::TypeDescriptor::element_type

If this descriptor represents an array, sequence, or string type, this property indicates the element type of the collection.

It must not be nil for the descriptor to be consistent. If this descriptor represents a map type, this property indicates the value element type of the map. It must not be nil for the descriptor to be consistent.

If this descriptor represents a bit set type, this property must indicate a Boolean type for the descriptor to be consistent.

If this descriptor represents any other kind of type, this property must be nil for the descriptor to be consistent.

12.12.4.4 class DynamicType* DDS::TypeDescriptor::key_element_type

If this descriptor represents a map type, this property indicates the value element type of the map.

It must not be nil for the descriptor to be consistent.

If this descriptor represents any other kind of type, this property must be nil for the descriptor to be consistent.

Chapter 13

CoreDX DDS Transports

13.1 CoreDX_LmtTransport Struct Reference

An instance of a Transport that can be added to a DomainParticipant.

Related Functions

(Note that these are not member functions.)

- **DDS_ReturnCode_t CoreDX_LmtTransport_get_default_config** (**CoreDX_LmtTransportConfig** ∗config)
- **DDS_ReturnCode_t CoreDX_LmtTransport_get_env_config** (**CoreDX_LmtTransportConfig** ∗config)
- **DDS_ReturnCode_t CoreDX_LmtTransport_clear_config** (**CoreDX_LmtTransportConfig** ∗config)
- CoreDX_Transport ∗ **CoreDX_LmtTransport_create_transport** (**CoreDX_LmtTransportConfig** ∗config)

13.1.1 Detailed Description

An instance of a 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

DDS_DomainParticipant_add_transport()

13.1.2 Friends And Related Function Documentation

13.1.2.1 DDS_ReturnCode_t CoreDX_LmtTransport_clear_config (CoreDX_LmtTransportConfig ∗ *config*) [related]

Clears any dynamic memory allocated during initialization of the **CoreDX_LmtTransportConfig** (p. 196) object.

13.1.2.2 CoreDX_Transport * CoreDX_LmtTransport_create_transport (CoreDX_LmtTransportConfig * *config*) [related]

Allocate and initialize a **CoreDX_LmtTransport** (p. 195) object instance. The returned CoreDX_Transport object can be added to a single DomainParticipant.

Note

The DomainParticipant takes ownership of the provided transport; when the DomainParticipant is deleted, it will release the resources of all added transports.

13.1.2.3 DDS_ReturnCode_t CoreDX_LmtTransport_get_default_config (CoreDX_LmtTransportConfig * *config*) [related]

Initialize the LmtTransportConfig object with default values. Currently assigned values may be overwritten by defaults.

13.1.2.4 DDS_ReturnCode_t CoreDX_LmtTransport_get_env_config (CoreDX_LmtTransportConfig * *config*) [related]

Query for environment variables that impact lmt transport configuration. Load the values (if any) into the LmtTransportConfig object. Currently assigned values may be overwritten by values derived from environment variables.

13.2 CoreDX_LmtTransportConfig Struct Reference

Structure that holds LMT Transport configuration items.

Public Attributes

- int **so_sndbuf**
- int **so_rcvbuf**
- unsigned int **max_tx_size**
- unsigned int **max_rx_buf_size**
- unsigned int **debug_flags**

13.2.1 Detailed Description

Structure that holds LMT Transport configuration items.

See also

DomainParticipant_add_transport()

13.2.2 Member Data Documentation

13.2.2.1 unsigned int CoreDX_LmtTransportConfig::debug_flags

adjust the debug output of the transport

13.2.2.2 unsigned int CoreDX_LmtTransportConfig::max_rx_buf_size

limit size of RX buffer (per connection)

13.2.2.3 unsigned int CoreDX_LmtTransportConfig::max_tx_size

largest LMT packet size we transmit (default: 64K, max: 64K)

13.2.2.4 int CoreDX_LmtTransportConfig::so_rcvbuf

size in bytes for socket RCVBUF

13.2.2.5 int CoreDX_LmtTransportConfig::so_sndbuf

size in bytes for socket SNDBUF

13.3 CoreDX_TcpTransport Struct Reference

An instance of a Transport that can be added to a DomainParticipant.

Related Functions

(Note that these are not member functions.)

- **DDS_ReturnCode_t CoreDX_TcpTransport_get_default_config** (**CoreDX_TcpTransportConfig** ∗config)
- **DDS_ReturnCode_t CoreDX_TcpTransport_get_env_config** (**CoreDX_TcpTransportConfig** ∗config)
- **DDS_ReturnCode_t CoreDX_TcpTransport_clear_config** (**CoreDX_TcpTransportConfig** ∗config)
- CoreDX_Transport ∗ **CoreDX_TcpTransport_create_transport** (**CoreDX_TcpTransportConfig** ∗config)

13.3.1 Detailed Description

An instance of a 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

DDS_DomainParticipant_add_transport()

13.3.2 Friends And Related Function Documentation

13.3.2.1 DDS_ReturnCode_t CoreDX_TcpTransport_clear_config (CoreDX_TcpTransportConfig ∗ *config*) [related]

Clears any dynamic memory allocated during initialization of the **CoreDX_TcpTransportConfig** (p. 198) object.

13.3.2.2 CoreDX_Transport * CoreDX_TcpTransport_create_transport (CoreDX_TcpTransportConfig * *config*) [related]

Allocate and initialize a **CoreDX_TcpTransport** (p. 197) object instance. The returned CoreDX_Transport object can be added to a single DomainParticipant.

Note

The DomainParticipant takes ownership of the provided transport; when the DomainParticipant is deleted, it will release the resources of all added transports.

13.3.2.3 DDS_ReturnCode_t CoreDX_TcpTransport_get_default_config (CoreDX_TcpTransportConfig * *config*) [related]

Initialize the TcpTransportConfig object with default values. Currently assigned values may be overwritten by defaults.

13.3.2.4 DDS_ReturnCode_t CoreDX_TcpTransport_get_env_config (CoreDX_TcpTransportConfig * *config*) [related]

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.

13.4 CoreDX_TcpTransportConfig Struct Reference

Structure that holds TCP Transport configuration items.

Public Attributes

- short **participant_index**
- CoreDX_IpTransportInterfaceSeq **interfaces**
- unsigned char **dynamic_interfaces**
- int **tx_max_packet_size**
- unsigned int **debug_flags**

13.4.1 Detailed Description

Structure that holds TCP Transport configuration items.

See also

DomainParticipant::add_transport(Transport) add_transport()

13.4.2 Member Data Documentation

13.4.2.1 unsigned int CoreDX_TcpTransportConfig::debug_flags

adjust the debug output of the transport

13.4.2.2 unsigned char CoreDX_TcpTransportConfig::dynamic_interfaces

detect and handle changes to interface addresses

13.4.2.3 CoreDX_IpTransportInterfaceSeq CoreDX_TcpTransportConfig::interfaces

default: empty -> use all available interfaces

13.4.2.4 short CoreDX_TcpTransportConfig::participant_index

-1: auto detect; else force (may fail if another participant is using the ports (can't exceed 120)

13.4.2.5 int CoreDX_TcpTransportConfig::tx_max_packet_size

default: 64K (to match the udp limit)

13.5 CoreDX_UdpTransport Struct Reference

An instance of a Transport that can be added to a DomainParticipant.

Related Functions

(Note that these are not member functions.)

- **DDS_ReturnCode_t CoreDX_UdpTransport_get_default_config** (**CoreDX_UdpTransportConfig** ∗config)
- **DDS_ReturnCode_t CoreDX_UdpTransport_get_env_config** (**CoreDX_UdpTransportConfig** ∗config)
- **DDS_ReturnCode_t CoreDX_UdpTransport_clear_config** (**CoreDX_UdpTransportConfig** ∗config)
- CoreDX_Transport ∗ **CoreDX_UdpTransport_create_transport** (**CoreDX_UdpTransportConfig** ∗config)

13.5.1 Detailed 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

DDS_DomainParticipant_add_transport()

13.5.2 Friends And Related Function Documentation

13.5.2.1 DDS_ReturnCode_t CoreDX_UdpTransport_clear_config (CoreDX_UdpTransportConfig ∗ *config*) [related]

Clears any dynamic memory allocated during initialization of the **CoreDX_UdpTransportConfig** (p. 200) object.

13.5.2.2 CoreDX_Transport * CoreDX_UdpTransport_create_transport (CoreDX_UdpTransportConfig * *config*) [related]

Allocate and initialize a **CoreDX_UdpTransport** (p. 199) object instance. The returned CoreDX_Transport object can be added to a single DomainParticipant.

Note

The DomainParticipant takes ownership of the provided transport; when the DomainParticipant is deleted, it will release the resources of all added transports.

13.5.2.3 DDS_ReturnCode_t CoreDX_UdpTransport_get_default_config (CoreDX_UdpTransportConfig * *config*) [related]

Initialize the UdpTransportConfig object with default values. Currently assigned values may be overwritten by defaults.

13.5.2.4 DDS_ReturnCode_t CoreDX_UdpTransport_get_env_config (CoreDX_UdpTransportConfig * *config*) [related]

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.

13.6 CoreDX_UdpTransportConfig Struct Reference

Structure that holds UDP Transport configuration items.

Public Attributes

- short **participant_index**
- unsigned char **use_ipv4**
- unsigned char **use_ipv6**
- CoreDX_IpTransportInterfaceSeq **interfaces**
- unsigned char **dynamic_interfaces**
- int **rx_init_buffer_size**
- int **rx_max_buffer_size**
- int **tx_max_packet_size**
- int **so_rcvbuf**
- int **so_sndbuf**
- unsigned char **meta_multicast_address_v4** [4]
- unsigned char **user_multicast_address_v4** [4]
- unsigned char **meta_multicast_address_v6** [16]
- unsigned char **user_multicast_address_v6** [16]
- unsigned char **multicast_ttl**
- unsigned char **tx_meta_multicast**
- unsigned char **tx_meta_unicast**
- unsigned char **rx_meta_multicast**
- unsigned char **rx_user_multicast**
- unsigned char **advertise_meta_multicast**

- unsigned char **advertise_user_multicast**
- unsigned char **broadcast_address** [4]
- unsigned char **do_meta_broadcast**
- unsigned int **debug_flags**

13.6.1 Detailed Description

Structure that holds UDP Transport configuration items.

See also

DomainParticipant_add_transport()

13.6.2 Member Data Documentation

13.6.2.1 unsigned char CoreDX_UdpTransportConfig::advertise_meta_multicast

advertise we can RX META MULTICAST

13.6.2.2 unsigned char CoreDX_UdpTransportConfig::advertise_user_multicast

advertise we can RX USER MULTICAST

13.6.2.3 unsigned char CoreDX_UdpTransportConfig::broadcast_address[4]

default: 255.255.255.255

13.6.2.4 unsigned int CoreDX_UdpTransportConfig::debug_flags

adjust the debug output from the transport

13.6.2.5 unsigned char CoreDX_UdpTransportConfig::do_meta_broadcast

enable broadcast of META (DPD discovery) data default: 0 (off)

13.6.2.6 unsigned char CoreDX_UdpTransportConfig::dynamic_interfaces

detect and handle changes to interface addresses

13.6.2.7 CoreDX_IpTransportInterfaceSeq CoreDX_UdpTransportConfig::interfaces

default: empty -> use all available interfaces

13.6.2.8 unsigned char CoreDX_UdpTransportConfig::meta_multicast_address_v4[4]

default: [239 255 0 1] per the standard

13.6.2.9 unsigned char CoreDX_UdpTransportConfig::meta_multicast_address_v6[16]

default: [ff03:0000:0000:0000:0000:efff:0001]

13.6.2.10 unsigned char CoreDX_UdpTransportConfig::multicast_ttl

default: 1 (0: disable all MCAST TX)

13.6.2.11 short CoreDX_UdpTransportConfig::participant_index

-1: auto detect; else force (may fail if another participant is using the ports (can't exceed 120)

13.6.2.12 int CoreDX_UdpTransportConfig::rx_init_buffer_size

initial size of data buffer

13.6.2.13 int CoreDX_UdpTransportConfig::rx_max_buffer_size

maximum size of data buffer

13.6.2.14 unsigned char CoreDX_UdpTransportConfig::rx_meta_multicast

enable META MULTICAST (discovery) RX

13.6.2.15 unsigned char CoreDX_UdpTransportConfig::rx_user_multicast

enable USER MULTICAST (data) RX

13.6.2.16 int CoreDX_UdpTransportConfig::so_rcvbuf

socket RCVBUF size (set to -1 to use OS default)

13.6.2.17 int CoreDX_UdpTransportConfig::so_sndbuf

socket SNDBUF size (set to -1 to use OS default)

13.6.2.18 int CoreDX_UdpTransportConfig::tx_max_packet_size

default: 64K (udp limit)

13.6.2.19 unsigned char CoreDX_UdpTransportConfig::tx_meta_multicast

enable META MULTICAST (discovery) TX

13.6.2.20 unsigned char CoreDX_UdpTransportConfig::tx_meta_unicast

enable META UNICAST (discovery) TX

13.6.2.21 unsigned char CoreDX_UdpTransportConfig::use_ipv4

Support IPv4 communications (default ON (1))

13.6.2.22 unsigned char CoreDX_UdpTransportConfig::use_ipv6

Support IPv4 communications (default OFF(0))

13.6.2.23 unsigned char CoreDX_UdpTransportConfig::user_multicast_address_v4[4]

default: [239 255 0 1] per the standard

13.6.2.24 unsigned char CoreDX_UdpTransportConfig::user_multicast_address_v6[16]

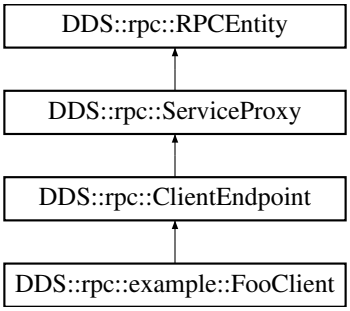
default: [ff03:0000:0000:0000:0000:efff:0001]

Chapter 14

CoreDX DDS RPC over DDS API

14.1 DDS::rpc::ClientEndpoint Class Reference

Inheritance diagram for DDS::rpc::ClientEndpoint:



Public Member Functions

- **ClientEndpoint** ()

  Default constructor.

- template<class TReq >
  dds_type_traits< TReq >::**DataWriter get_request_datawriter** () const

  *Access the **DataWriter** (p. 37) used to send TReq requests.*

- template<class TRep >
  dds_type_traits< TRep >::**DataReader get_reply_datareader** () const

  *Access the **DataReader** (p. 26) used to receive TRep replies.*

- **DDS::rpc::ClientParams get_client_params** ()

  *Access the **ClientParams** (p. 206) defining the configuration of this **ClientEndpoint** (p. 205).*

14.1.1 Detailed Description

A **ClientEndpoint** (p. 205) provides functions to obtain the underlying DDS entities at the client side. **ClientEndpoint** (p. 205) inherits from **ServiceProxy** (p. 228). A **ClientEndpoint** (p. 205) shall not be instantiated directly; it can be obtained from a Client object.

14.2 DDS::rpc::ClientParams Class Reference

Used to pass configuration parameters when constructing a Client.

Public Member Functions

- **ClientParams** ()

  Default constructor.
- **ClientParams** (const **ClientParams** &other)

  Copy constructor.
- **ClientParams** & **operator=** (const **ClientParams** &that)

  Assignment operator.
- **ClientParams** & **service_name** (const std::string &service_name)

  *Assign the **service_name** configured in this instance of **ClientParams** (p. 206).*
- **ClientParams** & **instance_name** (const std::string &instance_name)

  *Assign the **instance_name** configured in this instance of **ClientParams** (p. 206).*
- **ClientParams** & **request_topic_name** (const std::string &req_topic)

  *Assign the **request_topic_name** configured in this instance of **ClientParams** (p. 206).*
- **ClientParams** & **reply_topic_name** (const std::string &rep_topic)

  *Assign the **reply_topic_name** configured in this instance of **ClientParams** (p. 206).*
- **ClientParams** & **datawriter_qos** (const dds_entity_traits::DataWriterQos &qos)

  *Assign the **DataWriterQos** (p. 76) configured in this instance of **ClientParams** (p. 206).*
- **ClientParams** & **datareader_qos** (const dds_entity_traits::DataReaderQos &qos)

  *Assign the **DataReaderQos** (p. 75) configured in this instance of **ClientParams** (p. 206).*
- **ClientParams** & **publisher** (dds_entity_traits::Publisher publisher)

  *Assign the **Publisher** (p. 57) configured in this instance of **ClientParams** (p. 206).*
- **ClientParams** & **subscriber** (dds_entity_traits::Subscriber subscriber)

  *Assign the **Subscriber** (p. 65) configured in this instance of **ClientParams** (p. 206).*
- **ClientParams** & **domain_participant** (dds_entity_traits::DomainParticipant part)

  *Assign the **DomainParticipant** (p. 44) configured in this instance of **ClientParams** (p. 206).*
- const std::string **service_name** () const

  *Access the **service_name** configured in this instance of **ClientParams** (p. 206).*
- const std::string **instance_name** () const

  *Access the **instance_name** configured in this instance of **ClientParams** (p. 206).*
- const std::string **request_topic_name** () const

  *Access the **request_topic_name** configured in this instance of **ClientParams** (p. 206).*
- const std::string **reply_topic_name** () const

  *Access the **reply_topic_name** configured in this instance of **ClientParams** (p. 206).*
- const dds_entity_traits::DataWriterQos **datawriter_qos** () const

  *Access the **DataWriterQos** (p. 76) configured in this instance of **ClientParams** (p. 206).*
- const dds_entity_traits::DataReaderQos **datareader_qos** () const

  *Access the **DataReaderQos** (p. 75) configured in this instance of **ClientParams** (p. 206).*
- dds_entity_traits::Publisher **publisher** () const

  *Access the **Publisher** (p. 57) configured in this instance of **ClientParams** (p. 206).*
- dds_entity_traits::Subscriber **subscriber** () const

  *Access the **Subscriber** (p. 65) configured in this instance of **ClientParams** (p. 206).*
- dds_entity_traits::DomainParticipant **domain_participant** () const

  *Access the **DomainParticipant** (p. 44) configured in this instance of **ClientParams** (p. 206).*

14.2.1 Detailed Description

Used to pass configuration parameters when constructing a Client.

ClientParams (p. 206) 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.

14.3 DDS::rpc::ListenerBase Class Reference

Basis for all rpc related listener classes.

Inheritance diagram for DDS::rpc::ListenerBase:

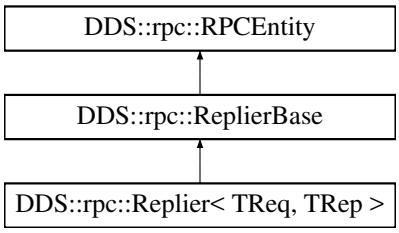
14.3.1 Detailed Description

Basis for all rpc related listener classes.

14.4 DDS::rpc::Replier< TReq, TRep > Class Template Reference

A replier receives requests and send replies.

Inheritance diagram for DDS::rpc::Replier< TReq, TRep >:



Public Member Functions

- **Replier** ()

  Default constructor.
- **Replier** (const **ReplierParams** &params)

  *Constructor that accepts **ReplierParams** (p. 212) configuration.*
- virtual ~**Replier** ()

  Destructor.
- void **send_reply** (**WriteSample**< TRep > &reply, const **DDS::SampleIdentity_t** &related_request_id)

  Sends the provided 'reply' associated with 'related_request_id'.
- void **send_reply** (**WriteSampleRef**< TRep > &reply, const **DDS::SampleIdentity_t** &related_request_id)

  Sends the provided 'reply' associated with 'related_request_id'.
- bool **receive_request** (**Sample**< TReq > &request, const **DDS::Duration_t** &max_wait)

  Attempt to access the next request.

- bool **receive_request** (**SampleRef**< TReq > request, const **DDS::Duration_t** &max_wait)

  Attempt to access the next request.
- **LoanedSamplesType receive_requests** (const **DDS::Duration_t** &max_wait)

  Attempt to access any available requests.
- **LoanedSamplesType receive_requests** (uint32_t min_request_count, uint32_t max_request_count, const **DDS::Duration_t** &max_wait)

  Attempt to access up to 'max_request_count' requests.
- bool **wait_for_requests** (const **DDS::Duration_t** &max_wait)

  Block until at least one request is available.
- bool **wait_for_requests** (uint32_t min_count, const **DDS::Duration_t** &max_wait)

  Block until at least 'min_count requests available.
- bool **take_request** (**Sample**< TReq > &request)

  Attempt to access the next request.
- bool **take_request** (**SampleRef**< TReq > request)

  Attempt to access the next request.
- **LoanedSamplesType take_requests** (int max_samples)

  Attempt to access up to 'max_samples' requests.
- bool **read_request** (**Sample**< TReq > &request)

  Attempt to access the next request.
- bool **read_request** (**SampleRef**< TReq > request)

  Attempt to access the next request.
- **LoanedSamplesType read_requests** (int max_samples)

  Attempt to access up to 'max_samples' requests.
- const **ReplierParams** & **get_replier_params** () const

  *Access the **ReplierParams** (p. 212) values that define the configuration of the **Replier** (p. 207).*
- bool **receive_nondata_samples** (bool enable)

  Toggle the underlying entities to deliver non-data samples to the application.
- RequestDataReader **get_request_datareader** () const

  *Access the underlying **DataReader** (p. 26) used to receive Requests.*
- ReplyDataWriter **get_reply_datawriter** () const

  *Access the underlying **DataWriter** (p. 37) used to send Replies.*

14.4.1 Detailed Description

template<typename TReq, typename TRep>
class DDS::rpc::Replier< TReq, TRep >

A replier receives requests and send replies.

An instance of **Replier** (p. 207) is configured at the time of construction using **ReplierParams** (p. 212), which is a container of configuration parameters such as domain participant, QoS, listeners and more.

Replier (p. 207) allows listener-based and polling-based reception of requests.

SimpleReplierListener (p. 230) and **ReplierListener** (p. 212) interfaces enable call-back based notification when a request is available. On the other hand, **Replier** (p. 207) provides functions to allow polling reception of requests.

14.4.2 Constructor & Destructor Documentation

14.4.2.1 template<typename TReq, typename TRep> DDS::rpc::Replier< TReq, TRep >::Replier ( )

Default constructor.

Without any configuration, the **Replier** (p. 207) will construct DDS Entities as required in domain '0'.

14.4.3 Member Function Documentation

14.4.3.1 template<typename TReq, typename TRep> bool DDS::rpc::Replier< TReq, TRep >::read_request (Sample< TReq > & *request*)

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

14.4.3.2 template<typename TReq, typename TRep> bool DDS::rpc::Replier< TReq, TRep >::read_request (SampleRef< TReq > *request*)

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

14.4.3.3 template<typename TReq, typename TRep> LoanedSamplesType DDS::rpc::Replier< TReq, TRep >::read_requests (int *max_samples*)

Attempt to access up to 'max_samples' requests.

This call may block if no requests are available.

Return values

collection	of requests

14.4.3.4 template<typename TReq, typename TRep> bool DDS::rpc::Replier< TReq, TRep >::receive_nondata_samples (bool *enable*)

Toggle the underlying entities to deliver non-data samples to the application.

Not Yet Supported

14.4.3.5 template<typename TReq, typename TRep> bool DDS::rpc::Replier< TReq, TRep >::receive_request (Sample< TReq > & *request,* const DDS::Duration_t & *max_wait*)

Attempt to access the next request.

This may block for up to 'max_wait' if no request is immediately available.

14.4.3.6 template<typename TReq, typename TRep> bool DDS::rpc::Replier< TReq, TRep >::receive_request (SampleRef< TReq > *request,* const DDS::Duration_t & *max_wait*)

Attempt to access the next request.

This may block for up to 'max_wait' if no request is immediately available.

14.4.3.7 template<typename TReq, typename TRep> LoanedSamplesType DDS::rpc::Replier< TReq, TRep >::receive_requests (const DDS::Duration_t & *max_wait*)

Attempt to access any available requests.

This may block for up to 'max_wait' if no requests are immediately available.

14.4.3.8 template<typename TReq, typename TRep> LoanedSamplesType DDS::rpc::Replier< TReq, TRep >::receive_requests (uint32_t *min_request_count,* uint32_t *max_request_count,* const DDS::Duration_t & *max_wait*)

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.

14.4.3.9 template<typename TReq, typename TRep> bool DDS::rpc::Replier< TReq, TRep >::take_request (Sample< TReq > & *request*)

Attempt to access the next request.

This call will not block.

Return values

true	if able to access a request.
false	otherwise

14.4.3.10 template<typename TReq, typename TRep> bool DDS::rpc::Replier< TReq, TRep >::take_request (SampleRef< TReq > *request*)

Attempt to access the next request.

This call will not block.

Return values

true	if able to access a request.
false	otherwise

14.4.3.11 template<typename TReq, typename TRep> LoanedSamplesType DDS::rpc::Replier< TReq, TRep >::take_requests (int *max_samples*)

Attempt to access up to 'max_samples' requests.

This call will not block.

Return values

collection	of requests (possibly empty)

14.4.3.12 template<typename TReq, typename TRep> bool DDS::rpc::Replier< TReq, TRep >::wait_for_requests (const DDS::Duration_t & *max_wait*)

Block until at least one request is available.

This may block for up to 'max_wait'.

14.4.3.13 template<typename TReq, typename TRep> bool DDS::rpc::Replier< TReq, TRep >::wait_for_requests (uint32_t *min_count,* const DDS::Duration_t & *max_wait*)

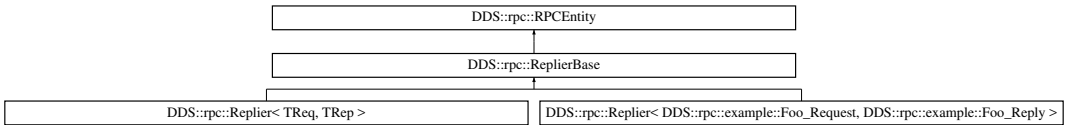
Block until at least 'min_count requests available.

This may block for up to 'max_wait'.

14.5 DDS::rpc::ReplierBase Class Reference

Provides a generic basis for all **Replier** (p. 207) objects.

Inheritance diagram for DDS::rpc::ReplierBase:



Additional Inherited Members

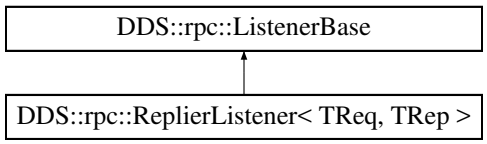
14.5.1 Detailed Description

Provides a generic basis for all **Replier** (p. 207) objects.

14.6 DDS::rpc::ReplierListener< TReq, TRep > Class Template Reference

ReplierLister can be installed on a **Replier** (p. 207).

Inheritance diagram for DDS::rpc::ReplierListener< TReq, TRep >:



Public Member Functions

- virtual void **on_request_available** (**Replier**< TReq, TRep > &)=0

  *Invoked when the **Replier** (p. 207) has one or more requests pending.*

14.6.1 Detailed Description

template<class TReq, class TRep>
class DDS::rpc::ReplierListener< TReq, TRep >

ReplierLister can be installed on a **Replier** (p. 207).

ReplierListener<TReq, TRep> interface is used to provide an asynchronous request listener for a **Replier** (p. 207). It is passed to the **Replier** (p. 207) constructor through **ReplierParams** (p. 212). It extends **ListenerBase** (p. 207) interface and enables asynchronous processing of the requests. The callback is provided the **Replier** (p. 207) object and returns void.

See also

ReplierParams (p. 212)

14.6.2 Member Function Documentation

14.6.2.1 template<class TReq, class TRep> virtual void DDS::rpc::ReplierListener< TReq, TRep >::on_request_available (Replier< TReq, TRep > &) [pure virtual]

Invoked when the **Replier** (p. 207) has one or more requests pending.

The method can process the available requests and send replies as required.

14.7 DDS::rpc::ReplierParams Class Reference

Used to pass configuration parameters when constructing a **Replier** (p. 207).

Public Member Functions

- **ReplierParams** ()

Default constructor.

- **ReplierParams** (const **ReplierParams** &other)

  Copy constructor.
- ~**ReplierParams** ()

  Destructor.
- **ReplierParams** & **operator=** (const **ReplierParams** &that)

  Assignment operator.
- template<class TReq , class TRep >
  ReplierParams & **simple_replier_listener** (**SimpleReplierListener**< TReq, TRep > *listener)

  *Assign the **SimpleReplierListener** (p. 230) configured in this instance of **ReplierParams** (p. 212).*
- template<class TReq , class TRep >
  ReplierParams & **replier_listener** (**ReplierListener**< TReq, TRep > *listener)

  *Assign the **ReplierListener** (p. 212) configured in this instance of **ReplierParams** (p. 212).*
- **ReplierParams** & **domain_participant** (dds_entity_traits::DomainParticipant participant)

  *Assign the **DomainParticipant** (p. 44) configured in this instance of **ReplierParams** (p. 212).*
- **ReplierParams** & **service_name** (const std::string &service_name)

  *Assign the **service_name** configured in this instance of **ReplierParams** (p. 212).*
- **ReplierParams** & **instance_name** (const std::string &instance_name)

  *Assign the **instance_name** configured in this instance of **ReplierParams** (p. 212).*
- **ReplierParams** & **request_topic_name** (const std::string &req_topic)

  *Assign the **request_topic_name** configured in this instance of **ReplierParams** (p. 212).*
- **ReplierParams** & **reply_topic_name** (const std::string &rep_topic)

  *Assign the **reply_topic_name** configured in this instance of **ReplierParams** (p. 212).*
- **ReplierParams** & **datawriter_qos** (dds_entity_traits::DataWriterQos qos)

  *Assign the **DataWriterQos** (p. 76) configured in this instance of **ReplierParams** (p. 212).*
- **ReplierParams** & **datareader_qos** (dds_entity_traits::DataReaderQos qos)

  *Assign the **DataReaderQos** (p. 75) configured in this instance of **ReplierParams** (p. 212).*
- **ReplierParams** & **publisher** (dds_entity_traits::Publisher publisher)

  *Assign the **Publisher** (p. 57) configured in this instance of **ReplierParams** (p. 212).*
- **ReplierParams** & **subscriber** (dds_entity_traits::Subscriber subscriber)

  *Assign the **Subscriber** (p. 65) configured in this instance of **ReplierParams** (p. 212).*
- dds_entity_traits::DomainParticipant **domain_participant** () const

  *Access the **DomainParticipant** (p. 44) configured in this instance of **ReplierParams** (p. 212).*
- **ListenerBase** * **simple_replier_listener** () const

  *Access the **SimpleReplierListener** (p. 230) configured in this instance of ReplierParams.i.*
- **ListenerBase** * **replier_listener** () const

  *Access the **ReplierListener** (p. 212) configured in this instance of **ReplierParams** (p. 212).*
- std::string **service_name** () const

  *Access the service_name configured in this instance of **ReplierParams** (p. 212).*
- std::string **instance_name** () const

  *Access the instance_name configured in this instance of **ReplierParams** (p. 212).*
- std::string **request_topic_name** () const

  *Access the request_topic_name configured in this instance of **ReplierParams** (p. 212).*
- std::string **reply_topic_name** () const

  *Access the reply_topic_name configured in this instance of **ReplierParams** (p. 212).*
- dds_entity_traits::DataWriterQos **datawriter_qos** () const

  *Access the **DataWriterQos** (p. 76) configured in this instance of **ReplierParams** (p. 212).*

- dds_entity_traits::DataReaderQos **datareader_qos** () const

  *Access the **DataReaderQos** (p. 75) configured in this instance of **ReplierParams** (p. 212).*
- dds_entity_traits::Publisher **publisher** () const

  *Access the **Publisher** (p. 57) configured in this instance of **ReplierParams** (p. 212).*
- dds_entity_traits::Subscriber **subscriber** () const

  *Access the **Subscriber** (p. 65) configured in this instance of **ReplierParams** (p. 212).*

14.7.1 Detailed Description

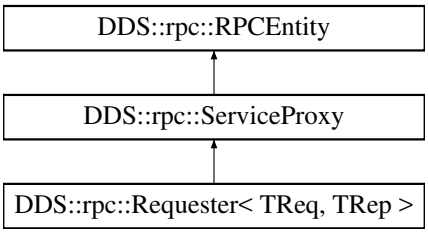
Used to pass configuration parameters when constructing a **Replier** (p. 207).

ReplierParams (p. 212) is a valuetype that serves as a container of configuration parameters of a **Replier** (p. 207). It is designed to mimic the named-parameters feature available in some programming languages, which improves readability.

14.8 DDS::rpc::Requester< TReq, TRep > Class Template Reference

A **Requester** (p. 214) sends requests and receives replies.

Inheritance diagram for DDS::rpc::Requester< TReq, TRep >:



Public Member Functions

- **Requester** ()

  Default constructor.
- **Requester** (const **RequesterParams** &params)

  *Constructor that accepts **RequesterParams** (p. 222) for configuration.*
- virtual ~**Requester** ()

  Destructor.
- void **send_request** (**WriteSample**< TReq > &request)

  Transmits a request to any matched instances of this service This method accepts a parameter of WriteSample< TReq > &.
- void **send_request** (**WriteSampleRef**< TReq > &wsref)

  Transmits a request to any matched instances of this service This method accepts a parameter of WriteSampleRef< TReq > ^&&.
- bool **receive_reply** (**Sample**< TRep > &reply, const **DDS::Duration_t** &timeout)

  Attempts to access a received reply.
- bool **receive_reply** (**SampleRef**< TRep > reply, const **DDS::Duration_t** &timeout)

  Attempts to access a received reply.
- bool **receive_reply** (**Sample**< TRep > &reply, const **DDS::SampleIdentity_t** &relatedRequestId)

Attempts to access a received reply.

- bool **receive_reply** (**SampleRef**< TRep > reply, const **DDS::SampleIdentity_t** &relatedRequestId)

  Attempts to access a received reply.
- **LoanedSamples**< TRep > **receive_replies** (const **DDS::Duration_t** &max_wait)

  Attempts to access multiple received replies.
- **LoanedSamples**< TRep > **receive_replies** (uint32_t min_count, uint32_t max_count, const **DDS::Duration_t** &max_wait)

  Attempts to access at least 'min_count' replies.
- bool **wait_for_replies** (uint32_t min_count, const **DDS::Duration_t** &max_wait)

  Blocks until at least 'min_count' replies are available.
- bool **wait_for_replies** (const **DDS::Duration_t** &max_wait)

  Blocks until a reply is available.
- bool **wait_for_replies** (uint32_t min_count, const **DDS::Duration_t** &max_wait, const **DDS::SampleIdentity_t** &related_request_id)

  Blocks until at least 'min_count' replies to 'related_request_id' are available.
- bool **take_reply** (**Sample**< TRep > &reply)

  Attempts to access a received reply.
- bool **take_reply** (**SampleRef**< TRep > reply)

  Attempts to access a received reply.
- bool **take_reply** (**Sample**< TRep > &reply, const **DDS::SampleIdentity_t** &related_request_id)

  Attempts to access a received reply.
- bool **take_reply** (**SampleRef**< TRep > reply, const **DDS::SampleIdentity_t** &related_request_id)

  Attempts to access a received reply.
- **LoanedSamples**< TRep > **take_replies** (uint32_t max_count)

  Attempts to access up to 'max_count' replies.
- **LoanedSamples**< TRep > **take_replies** (uint32_t max_count, const **DDS::SampleIdentity_t** &related_request_id)

  Attempts to access up to 'max_count' replies that match the 'related_request_id'.
- **LoanedSamples**< TRep > **take_replies** (const **DDS::SampleIdentity_t** &related_request_id)

  Attempts to access any replies that match the 'related_request_id'.
- bool **read_reply** (**Sample**< TRep > &reply)

  Attempts to access the next available reply.
- bool **read_reply** (**SampleRef**< TRep > reply)

  Attempts to access the next available reply.
- bool **read_reply** (**Sample**< TRep > &reply, const **DDS::SampleIdentity_t** &related_request_id)

  Attempts to access the next available reply that matches 'related_reqeust_id'.
- bool **read_reply** (**SampleRef**< TRep > reply, const **DDS::SampleIdentity_t** &related_request_id)

  Attempts to access the next available reply that matches 'related_reqeust_id'.
- **LoanedSamples**< TRep > **read_replies** (uint32_t max_count)

  Attempts to access up to 'max_count' replies.
- **LoanedSamples**< TRep > **read_replies** (uint32_t max_count, const **DDS::SampleIdentity_t** &related_request_id)

  Attempts to access up to 'max_count' replies that match 'related_request_id'.
- **LoanedSamples**< TRep > **read_replies** (const **DDS::SampleIdentity_t** &related_request_id)

  Attempts to access all available replies that match 'related_request_id'.
- bool **receive_nondata_samples** (bool enable)

  Toggle whether non-data samples are presented to the application.
- **RequesterParams get_requester_params** () const

Obtain the parameters used to configure this Reqeuster.

- RequestDataWriter **get_request_datawriter** () const

  *Obtain access to the underlying **DataWriter** (p. 37) used for sending Requests.*

- ReplyDataReader **get_reply_datareader** () const

  *Obtain access to the underlying **DataReader** (p. 26) used for receiving Replies.*

14.8.1 Detailed Description

template<typename TReq, typename TRep>
class DDS::rpc::Requester< TReq, TRep >

A **Requester** (p. 214) sends requests and receives replies.

An instance of a **Requester** (p. 214) is configured at the time of construction using **RequesterParams** (p. 222), which is a container of configuration parameters, such as domain participant, QoS, listeners and more.

Requester (p. 214) is inherently asynchronous as sending a request and receiving its corresponding reply (or replies) are separated. **Requester** (p. 214) allows listener-based, polling-based, and future-based reception of replies.

SimpleRequesterListener (p. 231) and **RequesterListener** (p. 221) interfaces enable callback-based notification when a reply is available. On the other hand, **Requester** (p. 214) 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.

14.8.2 Constructor & Destructor Documentation

14.8.2.1 template<typename TReq , typename TRep > DDS::rpc::Requester< TReq, TRep >::Requester ( )

Default constructor.

Without any configuration, the **Requester** (p. 214) will construct DDS Entities as required in domain '0'.

14.8.3 Member Function Documentation

14.8.3.1 template<typename TReq , typename TRep > LoanedSamples<TRep> DDS::rpc::Requester< TReq, TRep >::read_replies (uint32_t *max_count*)

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.

14.8.3.2 template<typename TReq , typename TRep > LoanedSamples<TRep> DDS::rpc::Requester< TReq, TRep >::read_replies (uint32_t *max_count,* const DDS::SampleIdentity_t & *related_request_id*)

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.

14.8.3.3 template<typename TReq , typename TRep > LoanedSamples<TRep> DDS::rpc::Requester< TReq, TRep >::read_replies (const DDS::SampleIdentity_t & *related_request_id*)

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.

14.8.3.4 template<typename TReq , typename TRep > bool DDS::rpc::Requester< TReq, TRep >::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'.

14.8.3.5 template<typename TReq , typename TRep > bool DDS::rpc::Requester< TReq, TRep >::read_reply (SampleRef< TRep > *reply*) [inline]

Attempts to access the next available reply.

This routine will block until a reply is available.

14.8.3.6 template<typename TReq , typename TRep > bool DDS::rpc::Requester< TReq, TRep >::read_reply (Sample< TRep > & *reply,* const DDS::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.

14.8.3.7 template<typename TReq , typename TRep > bool DDS::rpc::Requester< TReq, TRep >::read_reply (SampleRef< TRep > *reply,* const DDS::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.

14.8.3.8 template<typename TReq , typename TRep > bool DDS::rpc::Requester< TReq, TRep >::receive_nondata_samples (bool *enable*)

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.

14.8.3.9 template<typename TReq , typename TRep > LoanedSamples<TRep> DDS::rpc::Requester< TReq, TRep >::receive_replies (const DDS::Duration_t & *max_wait*)

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).

14.8.3.10 template<typename TReq , typename TRep > LoanedSamples<TRep> DDS::rpc::Requester< TReq, TRep >::receive_replies (uint32_t *min_count,* uint32_t *max_count,* const DDS::Duration_t & *max_wait*)

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).

14.8.3.11 template<typename TReq , typename TRep > bool DDS::rpc::Requester< TReq, TRep >::receive_reply (Sample< TRep > & *reply,* const DDS::Duration_t & *timeout*)

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).

14.8.3.12 template<typename TReq , typename TRep > bool DDS::rpc::Requester< TReq, TRep >::receive_reply (SampleRef< TRep > *reply,* const DDS::Duration_t & *timeout*)

Attempts to access a received reply.

This method accepts a parameter of SampleRef<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).

14.8.3.13 template<typename TReq , typename TRep > bool DDS::rpc::Requester< TReq, TRep >::receive_reply (Sample< TRep > & *reply,* const DDS::SampleIdentity_t & *relatedRequestId*)

Attempts to access a received reply.

This method accepts a parameter of Sample<TRep> & and a **SampleIdentity_t** (p. 133). 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).

14.8.3.14 template<typename TReq , typename TRep > bool DDS::rpc::Requester< TReq, TRep >::receive_reply (SampleRef< TRep > *reply,* const DDS::SampleIdentity_t & *relatedRequestId*)

Attempts to access a received reply.

This method accepts a parameter of SampleRef<TRep> & and a **SampleIdentity_t** (p. 133). 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).

14.8.3.15 template<typename TReq , typename TRep > LoanedSamples<TRep> DDS::rpc::Requester< TReq, TRep >::take_replies (uint32_t *max_count*)

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.

14.8.3.16 template<typename TReq , typename TRep > LoanedSamples<TRep> DDS::rpc::Requester< TReq, TRep >::take_replies (uint32_t *max_count,* const DDS::SampleIdentity_t & *related_request_id*)

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.

14.8.3.17 template<typename TReq , typename TRep > LoanedSamples<TRep> DDS::rpc::Requester< TReq, TRep >::take_replies (const DDS::SampleIdentity_t & *related_request_id*)

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.

14.8.3.18 template<typename TReq , typename TRep > bool DDS::rpc::Requester< TReq, TRep >::take_reply (Sample< TRep > & *reply*)

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).

14.8.3.19 template<typename TReq , typename TRep > bool DDS::rpc::Requester< TReq, TRep >::take_reply (SampleRef< TRep > *reply*)

Attempts to access a received reply.

This method accepts a parameter of SampleRef<TRep> & . The call will not block.

Return values

bool	indicating if a reply was found (true) or not (false).

14.8.3.20 template<typename TReq , typename TRep > bool DDS::rpc::Requester< TReq, TRep >::take_reply (Sample< TRep > & *reply,* const DDS::SampleIdentity_t & *related_request_id*)

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).

14.8.3.21 template<typename TReq , typename TRep > bool DDS::rpc::Requester< TReq, TRep >::take_reply (SampleRef< TRep > *reply,* const DDS::SampleIdentity_t & *related_request_id*)

Attempts to access a received reply.

This method accepts a parameter of SampleRef<TRep> & . The call will not block.

Return values

bool	indicating if a reply was found (true) or not (false).

14.8.3.22 template<typename TReq , typename TRep > bool DDS::rpc::Requester< TReq, TRep >::wait_for_replies (uint32_t *min_count,* const DDS::Duration_t & *max_wait*)

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).

14.8.3.23 template<typename TReq , typename TRep > bool DDS::rpc::Requester< TReq, TRep >::wait_for_replies (const DDS::Duration_t & *max_wait*)

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).

14.8.3.24 template<typename TReq , typename TRep > bool DDS::rpc::Requester< TReq, TRep >::wait_for_replies (uint32_t *min_count,* const DDS::Duration_t & *max_wait,* const DDS::SampleIdentity_t & *related_request_id*)

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'.

Not Yet Supported Currently, this routine will return if **at** least 'min_count' replies are available – it ignores 'related_request'.

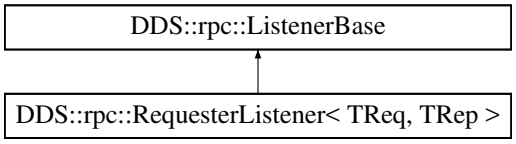
Return values

bool	indicating if replies were found (true) or not (false).

14.9 DDS::rpc::RequesterListener< TReq, TRep > Class Template Reference

RequesterListener (p. 221) can be installed on a **Requester** (p. 214).

Inheritance diagram for DDS::rpc::RequesterListener< TReq, TRep >:



Public Member Functions

- virtual void **on_reply_available** (**Requester**< TReq, TRep > &)=0

  Invoked when one or more replies are available.

14.9.1 Detailed Description

template<class TReq, class TRep>
class DDS::rpc::RequesterListener< TReq, TRep >

RequesterListener (p. 221) can be installed on a **Requester** (p. 214).

RequesterListener<TReq,TRep> interface is used to receive notification of reply arrival. It is passed to the **Requester** (p. 214) through **RequesterParams** (p. 222). It extends **ListenerBase** (p. 207) and enables processing of replies. The callback is provided the replier object and returns void.

See also

RequesterParams (p. 222)

14.9.2 Member Function Documentation

14.9.2.1 template<class TReq, class TRep> virtual void DDS::rpc::RequesterListener< TReq, TRep >::on_reply_available (Requester< TReq, TRep > &) [pure virtual]

Invoked when one or more replies are available.

Can process the replies as appropriate.

14.10 DDS::rpc::RequesterParams Class Reference

Used to pass configuration parameters when constructing a **Requester** (p. 214).

Public Member Functions

- **RequesterParams** ()

  Default constructor.
- ~**RequesterParams** ()

  Destructor.
- **RequesterParams** (const **RequesterParams** &other)

  Copy constructor.
- **RequesterParams** & **operator=** (const **RequesterParams** &that)

  Assignment operator.
- template<class TRep >
  RequesterParams & **simple_requester_listener** (**SimpleRequesterListener**< TRep > *listener)

  *Assign the 'simple' listener that will be installed in the **Requester** (p. 214).*
- template<class TReq , class TRep >
  RequesterParams & **requester_listener** (**RequesterListener**< TReq, TRep > *listener)

  *Assign the 'requester' listener that will be installed in the **Requester** (p. 214).*
- **RequesterParams** & **domain_participant** (dds_entity_traits::DomainParticipant participant)

  *Assign the **DomainParticipant** (p. 44) to use.*
- **RequesterParams** & **publisher** (dds_entity_traits::Publisher publisher)

  *Assign the **Publisher** (p. 57) to use.*
- **RequesterParams** & **subscriber** (dds_entity_traits::Subscriber subscriber)

  *Assign the **Subscriber** (p. 65) to use.*
- **RequesterParams** & **datawriter_qos** (dds_entity_traits::DataWriterQos qos)

  *Assign the **DataWriterQos** (p. 76) to use.*
- **RequesterParams** & **datareader_qos** (dds_entity_traits::DataReaderQos qos)

  *Assign the **DataReaderQos** (p. 75) to use.*
- **RequesterParams** & **service_name** (const std::string &name)

  Assign the service_name to use when creating the Request and Reply topic names.
- **RequesterParams** & **request_topic_name** (const std::string &name)

  Assign the 'request_topic_name' to use as the Request topic name.
- **RequesterParams** & **reply_topic_name** (const std::string &name)

Assign the 'reply_topic_name'.

- dds_entity_traits::DomainParticipant **domain_participant** () const

  *Access the **DomainParticipant** (p. 44) configured in this instance of **RequesterParams** (p. 222).*
- dds_entity_traits::Publisher **publisher** () const

  *Access the **Publisher** (p. 57) configured in this instance of **RequesterParams** (p. 222).*
- dds_entity_traits::Subscriber **subscriber** () const

  *Access the **Subscriber** (p. 65) configured in this instance of **RequesterParams** (p. 222).*
- dds_entity_traits::DataWriterQos **datawriter_qos** () const

  *Access the **DataWriterQos** (p. 76) configured in this instance of **RequesterParams** (p. 222).*
- dds_entity_traits::DataReaderQos **datareader_qos** () const

  *Access the **DataReaderQos** (p. 75) configured in this instance of **RequesterParams** (p. 222).*
- **ListenerBase** ∗ **simple_requester_listener** () const

  *Access the **SimpleRequesterListener** (p. 231) configured in this instance of **RequesterParams** (p. 222).*
- **ListenerBase** ∗ **requester_listener** () const

  *Access the **RequesterListener** (p. 221) configured in this instance of **RequesterParams** (p. 222).*
- std::string **service_name** () const

  *Access the service_name configured in this instance of **RequesterParams** (p. 222).*
- std::string **request_topic_name** () const

  *Access the request_topic_name configured in this instance of **RequesterParams** (p. 222).*
- std::string **reply_topic_name** () const

  *Access the reply_topic_name configured in this instance of **RequesterParams** (p. 222).*

14.10.1 Detailed Description

Used to pass configuration parameters when constructing a **Requester** (p. 214).

RequesterParams (p. 222) is a valuetype that serves as a container of configuration parameters of a **Requester** (p. 214). It is designed to mimic the named-parameters feature available in some programming languages, which improves readability.

14.10.2 Member Function Documentation

14.10.2.1 RequesterParams& DDS::rpc::RequesterParams::datareader_qos (dds_entity_traits::DataReaderQos *qos*)

Assign the **DataReaderQos** (p. 75) to use.

This QoS will be used when the **Requester** (p. 214) creates a **DataReader** (p. 26).

14.10.2.2 RequesterParams& DDS::rpc::RequesterParams::datawriter_qos (dds_entity_traits::DataWriterQos *qos*)

Assign the **DataWriterQos** (p. 76) to use.

This QoS will be used when the **Requester** (p. 214) creates a **DataWriter** (p. 37).

14.10.2.3 RequesterParams& DDS::rpc::RequesterParams::domain_participant (dds_entity_traits::DomainParticipant *participant*)

Assign the **DomainParticipant** (p. 44) to use.

The requester will use this **DomainParticipant** (p. 44) to create any required **Publisher** (p. 57), **Subscriber** (p. 65), **DataWriter** (p. 37) and **DataReader** (p. 26) entities.

14.10.2.4 RequesterParams& DDS::rpc::RequesterParams::publisher (dds_entity_traits::Publisher *publisher*)

Assign the **Publisher** (p. 57) to use.

The **Requester** (p. 214) will use this publisher to create any required **DataWriter** (p. 37) entities. If not provided the **Requester** (p. 214) will create its own **Publisher** (p. 57).

14.10.2.5 RequesterParams& DDS::rpc::RequesterParams::reply_topic_name (const std::string & *name*)

Assign the 'reply_topic_name'.

This overrides the default topic name generation.

14.10.2.6 RequesterParams& DDS::rpc::RequesterParams::request_topic_name (const std::string & *name*)

Assign the 'request_topic_name' to use as the Request topic name.

This overrides the default topic name generation.

14.10.2.7 template<class TReq , class TRep > RequesterParams& DDS::rpc::RequesterParams::requester_listener (RequesterListener< TReq, TRep > * *listener*)

Assign the 'requester' listener that will be installed in the **Requester** (p. 214).

Only one listener may be installed, either the SimpleListener or **RequesterListener** (p. 221).

14.10.2.8 RequesterParams& DDS::rpc::RequesterParams::service_name (const std::string & *name*)

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()** (p. 224) configuration item, the request topic is composed of <service_name>_Request. Similarly, if not overridden by **reply_topic_name()** (p. 224), the reply topic name is composed of <service_name>_Reply. If the **service_name()** (p. 224) item is left unspecified, then the string "Service" is used.

14.10.2.9 template<class TRep > RequesterParams& DDS::rpc::RequesterParams::simple_requester_listener (SimpleRequesterListener< TRep > * *listener*)

Assign the 'simple' listener that will be installed in the **Requester** (p. 214).

Only one listener may be installed, either the SimpleListener or **RequesterListener** (p. 221).

14.10.2.10 RequesterParams& DDS::rpc::RequesterParams::subscriber (dds_entity_traits::Subscriber *subscriber*)

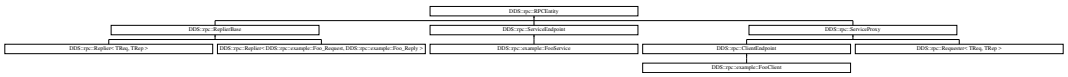
Assign the **Subscriber** (p. 65) to use.

The **Requester** (p. 214) will use this subscriber to create any requried **DataReader** (p. 26) entities. If not provided, the **Requester** (p. 214) will create its own **Subscriber** (p. 65).

14.11 DDS::rpc::RPCEntity Class Reference

RPCEntity (p. 225) is a base abstract class.

Inheritance diagram for DDS::rpc::RPCEntity:



Public Member Functions

- void **close** ()

  Release the underlying resources including any DDS entities.

- bool **is_null** () const

  Indicates if the entity has been 'closed'.

14.11.1 Detailed Description

RPCEntity (p. 225) is a base abstract class.

RPCEntity (p. 225) is the base abstract class extended by all the active RPC entities. It supports **close()** (p. 225) and **is_null()** (p. 225) operations.

14.11.2 Member Function Documentation

14.11.2.1 void DDS::rpc::RPCEntity::close ( )

Release the underlying resources including any DDS entities.

The **RPCEntity** (p. 225) will no longer be useful and will not participate in any communication.

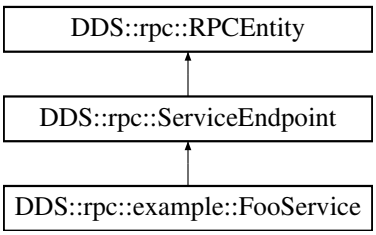
14.11.2.2 bool DDS::rpc::RPCEntity::is_null ( ) const

Indicates if the entity has been 'closed'.

If **is_null()** (p. 225) is true, then the underlying dds entities have been released and the **RPCEntity** (p. 225) is no longer useful for operations.

14.12 DDS::rpc::ServiceEndpoint Class Reference

Inheritance diagram for DDS::rpc::ServiceEndpoint:



Public Member Functions

- **ServiceEndpoint** ()

  Default constructor.
- template<class TReq >
  dds_type_traits< TReq >::**DataReader get_request_datareader** () const

  *Access the **DataReader** (p. 26) that is used to accept incoming TReq requests.*
- template<class TRep >
  dds_type_traits< TRep >::**DataWriter get_reply_datawriter** () const

  *Access the **DataWriter** (p. 37) that is used to send outgoing TRep replies.*
- void **pause** ()

  *Pause the operation of the **ServiceEndpoint** (p. 225).*
- void **resume** ()

  *Resume the operation of the **ServiceEndpoint** (p. 225).*
- **ServiceStatus status** () const

  *Access the current status of the **ServiceEndpoint** (p. 225).*
- **ServiceParams get_service_params** () const

  *Access the **ServiceParams** (p. 227) defining the configuration of this **ServiceEndpoint** (p. 225).*

14.12.1 Detailed Description

A **ServiceEndpoint** (p. 225) provides type-independent functions to manage a service (e.g., pause, resume, close, etc.) A **ServiceEndpoint** (p. 225) shall not be instantiated directly; it can be obtained from a Service object. **ServiceEndpoint** (p. 225) is a reference type. All 'Service' implementations derive from **ServiceEndpoint** (p. 225).

14.12.2 Member Function Documentation

14.12.2.1 void DDS::rpc::ServiceEndpoint::pause ( )

Pause the operation of the **ServiceEndpoint** (p. 225).

This causes the **ServiceEndpoint** (p. 225) to not invoke an attached listener.

14.12.2.2 void DDS::rpc::ServiceEndpoint::resume ( )

Resume the operation of the **ServiceEndpoint** (p. 225).

This causes the **ServiceEndpoint** (p. 225) to invoke any attached listeners upon reception of a request. In order to handle any requests received while the **ServiceEndpoint** (p. 225) was paused, the listener will be invoked as part of the **resume()** (p. 226) operation.

14.13 DDS::rpc::ServiceParams Class Reference

Used to pass configuration parameters when constructing a Service.

Public Member Functions

- **ServiceParams** ()

  Default constructor.
- **ServiceParams** (const **ServiceParams** &other)

  Copy constructor.
- **ServiceParams** & **operator=** (const **ServiceParams** &that)

  Assignment operator.
- **ServiceParams** & **service_name** (const std::string &service_name)

  *Assign the **service_name** configured in this instance of **ServiceParams** (p. 227).*
- **ServiceParams** & **instance_name** (const std::string &instance_name)

  *Assign the **instance_name** configured in this instance of **ServiceParams** (p. 227).*
- **ServiceParams** & **request_topic_name** (const std::string &req_topic)

  *Assign the **request_topic_name** configured in this instance of **ServiceParams** (p. 227).*
- **ServiceParams** & **reply_topic_name** (const std::string &rep_topic)

  *Assign the **reply_topic_name** configured in this instance of **ServiceParams** (p. 227).*
- **ServiceParams** & **datawriter_qos** (dds_entity_traits::DataWriterQos qos)

  *Assign the **DataWriterQos** (p. 76) configured in this instance of **ServiceParams** (p. 227).*
- **ServiceParams** & **datareader_qos** (dds_entity_traits::DataReaderQos qos)

  *Assign the **DataReaderQos** (p. 75) configured in this instance of **ServiceParams** (p. 227).*
- **ServiceParams** & **publisher** (dds_entity_traits::Publisher publisher)

  *Assign the **Publisher** (p. 57) configured in this instance of **ServiceParams** (p. 227).*
- **ServiceParams** & **subscriber** (dds_entity_traits::Subscriber subscriber)

  *Assign the **Subscriber** (p. 65) configured in this instance of **ServiceParams** (p. 227).*
- **ServiceParams** & **domain_participant** (dds_entity_traits::DomainParticipant part)

  *Assign the **DomainParticipant** (p. 44) configured in this instance of **ServiceParams** (p. 227).*
- std::string **service_name** () const

  *Access the **service_name** configured in this instance of **ServiceParams** (p. 227).*
- std::string **instance_name** () const

  *Access the **instance_name** configured in this instance of **ServiceParams** (p. 227).*
- std::string **request_topic_name** () const

  *Access the **request_topic_name** configured in this instance of **ServiceParams** (p. 227).*
- std::string **reply_topic_name** () const

  *Access the **reply_topic_name** configured in this instance of **ServiceParams** (p. 227).*
- dds_entity_traits::DataWriterQos **datawriter_qos** () const

  *Access the **DataWriterQos** (p. 76) configured in this instance of **ServiceParams** (p. 227).*
- dds_entity_traits::DataReaderQos **datareader_qos** () const

  *Access the **DataReaderQos** (p. 75) configured in this instance of **ServiceParams** (p. 227).*
- dds_entity_traits::Publisher **publisher** () const

  *Access the **Publisher** (p. 57) configured in this instance of **ServiceParams** (p. 227).*
- dds_entity_traits::Subscriber **subscriber** () const

  *Access the **Subscriber** (p. 65) configured in this instance of **ServiceParams** (p. 227).*
- dds_entity_traits::DomainParticipant **domain_participant** () const

  *Access the **DomainParticipant** (p. 44) configured in this instance of **ServiceParams** (p. 227).*

14.13.1 Detailed Description

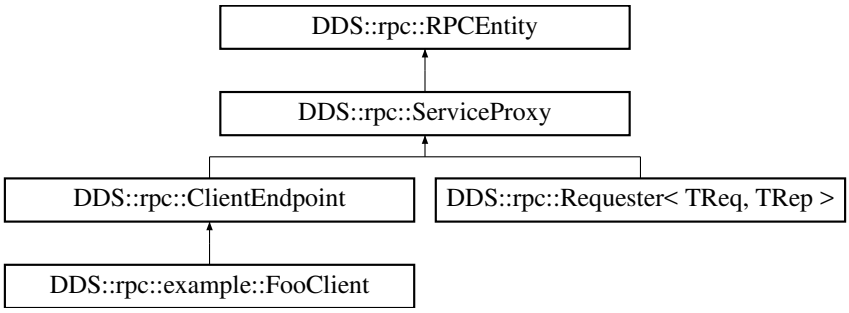
Used to pass configuration parameters when constructing a Service.

ServiceParams (p. 227) 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.

14.14 DDS::rpc::ServiceProxy Class Reference

ServiceProxy (p. 228).

Inheritance diagram for DDS::rpc::ServiceProxy:



Public Member Functions

- void **bind** (const std::string &instance_name)

  *Bind the **ServiceProxy** (p. 228) to service(s) with the provided 'instance_name'.*
- void **unbind** ()

  *Undo any previous '**bind()** (p. 229)' operation.*
- bool **is_bound** () const

  *Indicates if the **ServiceProxy** (p. 228) is currently bound to a particular instance_name.*
- std::string **get_bound_instance_name** () const

  *Provides the instance_name to which the **ServiceProxy** (p. 228) is currently bound.*
- std::vector< std::string > **get_discoverd_service_instances** () const

  *Access a list of instance_names that have been discovered by the **ServiceProxy** (p. 228).*
- std::vector< std::string > **get_discovered_service_instances** () const

  *Access a list of instance_names that have been discovered by the **ServiceProxy** (p. 228).*
- uint32_t **get_discovered_service_count** () const

  *Access the number of service instances that have been discovered by the **ServiceProxy** (p. 228).*
- void **wait_for_service** ()

  Block until any service is discovered.
- **ReturnCode_t wait_for_service** (const **DDS::Duration_t** &maxWait)

  Block up to 'maxWait' until any service is discovered.
- void **wait_for_service** (std::string instanceName)

  Wait for a service with matching 'instanceName' to be discovered.
- **ReturnCode_t wait_for_service** (const **DDS::Duration_t** &maxWait, std::string instanceName)

  Block up to 'maxWait' until a service matching 'instanceNmae' is discovered.
- void **wait_for_services** (uint32_t count)

Block until at least 'count' services have been discovered.

- **ReturnCode_t wait_for_services** (const **DDS::Duration_t** &maxWait, uint32_t count)

  Block until at least 'count' services have been discovered or 'maxWait' time elapses.

- void **wait_for_services** (const std::vector< std::string > &instanceNames)

  Block until discovering a service for each of the listed 'instanceNames'.

- **ReturnCode_t wait_for_services** (const **DDS::Duration_t** &maxWait, const std::vector< std::string > &instanceNames)

  Block until discovering a service for each of the listed 'instanceNames', or 'maxWait' elapses.

14.14.1 Detailed Description

ServiceProxy (p. 228).

ServiceProxy (p. 228) class defines type-independent operations for **Requester** (p. 214) and the Client. For example, binding to a specific instance, waiting for an instance to discover, closing the service, and more. **ServiceProxy** (p. 228) shall not be instantiated directly.

14.14.2 Member Function Documentation

14.14.2.1 void DDS::rpc::ServiceProxy::bind (const std::string & *instance_name*)

Bind the **ServiceProxy** (p. 228) to service(s) with the provided 'instance_name'.

The **ServiceProxy** (p. 228) will communicate only with services that have a matching 'instance_name'.

14.14.2.2 std::vector<std::string> DDS::rpc::ServiceProxy::get_discoverd_service_instances ( ) const

Access a list of instance_names that have been discovered by the **ServiceProxy** (p. 228).

The returned list is a snapshot in time of the currently known services. This routine is provided because the spec contains a typo in the method name (discoverd).

14.14.2.3 uint32_t DDS::rpc::ServiceProxy::get_discovered_service_count ( ) const

Access the number of service instances that have been discovered by the **ServiceProxy** (p. 228).

The returned number is a snapshot in time of the currently known services.

14.14.2.4 std::vector<std::string> DDS::rpc::ServiceProxy::get_discovered_service_instances ( ) const

Access a list of instance_names that have been discovered by the **ServiceProxy** (p. 228).

The returned list is a snapshot in time of the currently known services.

14.14.2.5 void DDS::rpc::ServiceProxy::unbind ( )

Undo any previous '**bind()** (p. 229)' operation.

After this, the **ServiceProxy** (p. 228) will communicate with any known services.

14.14.2.6 ReturnCode_t DDS::rpc::ServiceProxy::wait_for_service (const DDS::Duration_t & *maxWait*)

Block up to 'maxWait' until any service is discovered.

Return values

RETCODE_TIMEOUT	: If the maxWait time passes without discovering a service.

14.14.2.7 ReturnCode_t DDS::rpc::ServiceProxy::wait_for_service (const DDS::Duration_t & *maxWait,* std::string *instanceName*)

Block up to 'maxWait' until a service matching 'instanceNmae' is discovered.

Return values

RETCODE_TIMEOUT	: If the maxWait time passes without discovering a service.

14.14.2.8 ReturnCode_t DDS::rpc::ServiceProxy::wait_for_services (const DDS::Duration_t & *maxWait,* uint32_t *count*)

Block until at least 'count' services have been discovered or 'maxWait' time elapses.

Return values

RETCODE_TIMEOUT	: If the maxWait time passes without discovering 'count' services.

14.14.2.9 ReturnCode_t DDS::rpc::ServiceProxy::wait_for_services (const DDS::Duration_t & *maxWait,* const std::vector< std::string > & *instanceNames*)

Block until discovering a service for each of the listed 'instanceNames', or 'maxWait' elapses.

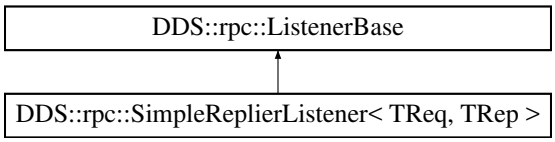
Return values

RETCODE_TIMEOUT	: If the maxWait time passes without discovering the specified services.

14.15 DDS::rpc::SimpleReplierListener< TReq, TRep > Class Template Reference

SimpleReplierListener (p. 230) can be installed on a **Replier** (p. 207).

Inheritance diagram for DDS::rpc::SimpleReplierListener< TReq, TRep >:



Public Member Functions

- virtual TRep * **process_request** (const **Sample**< TReq > &, const **DDS::SampleIdentity_t** &)=0

  *Invoked when the **Replier** (p. 207) has recieved a request.*

14.15.1 Detailed Description

template<class TReq, class TRep>
class DDS::rpc::SimpleReplierListener< TReq, TRep >

SimpleReplierListener (p. 230) can be installed on a **Replier** (p. 207).

SimpleReplierListener<TReq, TRep> is used to provide a synchronous request listener for a **Replier** (p. 207). **ReplierParams** (p. 212) is used to pass an instance of SimpleReplierListener<TReq, TRep>. It extends the **ListenerBase** (p. 207) abstract class and enables synchronous processing of the requests. The callback is provided the request object and must return a reply.

See also

ReplierParams (p. 212)

14.15.2 Member Function Documentation

14.15.2.1 template<class TReq, class TRep> virtual TRep* DDS::rpc::SimpleReplierListener< TReq, TRep >::process_request (const Sample< TReq > & , const DDS::SampleIdentity_t &) [pure virtual]

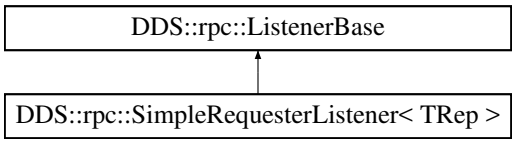
Invoked when the **Replier** (p. 207) has recieved a request.

Must initilaize and return a Reply.

14.16 DDS::rpc::SimpleRequesterListener< TRep > Class Template Reference

SimpleReqeusterListener can be installed on a **Requester** (p. 214).

Inheritance diagram for DDS::rpc::SimpleRequesterListener< TRep >:



Public Member Functions

- virtual void **process_reply** (const **Sample**< TRep > &, const **DDS::SampleIdentity_t** &)=0

  Invoked when a reply is available.

14.16.1 Detailed Description

template<class TRep>
class DDS::rpc::SimpleRequesterListener< TRep >

SimpleReqeusterListener can be installed on a **Requester** (p. 214).

SimpleRequesterListener<TRep> abstract class is used to receive notifications of arrival of replies. It is passed to the **Requester** (p. 214) through **RequesterParams** (p. 222). It extends **ListenerBase** (p. 207) and enables processing of replies. The callback is provided access to the reply sample and returns void.

See also

RequesterParams (p. 222)

14.16.2 Member Function Documentation

14.16.2.1 template<class TRep> virtual void DDS::rpc::SimpleRequesterListener< TRep >::process_reply (const Sample< TRep > & , const DDS::SampleIdentity_t &) [pure virtual]

Invoked when a reply is available.

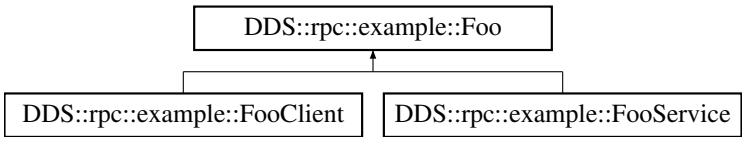
Must process the reply.

Chapter 15

CoreDX DDS RPC over DDS Example

15.1 DDS::rpc::example::Foo Struct Reference

Inheritance diagram for DDS::rpc::example::Foo:



Public Member Functions

- **Foo** ()
- virtual unsigned char **op1** (const int param)=0
- virtual int **op2** ()=0

15.1.1 Detailed Description

Data type generated to represent the methods of the **Foo** (p. 233) interface.

Associated IDL:

```
@service
 interface Foo (p. 233) {
    boolean op1( long param );
    long    op2( ) (p. 234);
};
```

15.1.2 Constructor & Destructor Documentation

15.1.2.1 DDS::rpc::example::Foo::Foo ( )

Constructor, Copy Constructor, Destructor, Assignment operator

15.1.3 Member Function Documentation

15.1.3.1 virtual unsigned char DDS::rpc::example::Foo::op1 (const int *param*) [pure virtual]

Foo::op1() (p. 234) pure virtual method. To be implemented by the Client and Service.

Implemented in **DDS::rpc::example::FooService** (p. 243), and **DDS::rpc::example::FooClient** (p. 242).

15.1.3.2 virtual int DDS::rpc::example::Foo::op2 ( ) [pure virtual]

Foo::op2() (p. 234) pure virtual method. To be implemented by the Client and Service.

Implemented in **DDS::rpc::example::FooService** (p. 243), and **DDS::rpc::example::FooClient** (p. 242).

15.2 DDS::rpc::example::Foo_Call Class Reference

Public Member Functions

- const struct **Foo_op1_In** & **op1** () const
- void **op1** (const struct **Foo_op1_In** &_v)
- const struct **Foo_op2_In** & **op2** () const
- void **op2** (const struct **Foo_op2_In** &_v)

15.2.1 Detailed Description

Data type generated to consolidate the parameters of any method of **Foo** (p. 233).

The data type is a union with one case for each operation in the **Foo** (p. 233) service. The requester must populate the specific union member that matches the method to be invoked.

15.2.2 Member Function Documentation

15.2.2.1 const struct Foo_op1_In& DDS::rpc::example::Foo_Call::op1 ( ) const [inline]

Access this to see the parameters to the method 'op1'.

15.2.2.2 void DDS::rpc::example::Foo_Call::op1 (const struct Foo_op1_In & *_v*) [inline]

Set this when invoking the method 'op1'.

15.2.2.3 const struct Foo_op2_In& DDS::rpc::example::Foo_Call::op2 ( ) const [inline]

Access this to see the parameters to the method 'op2'.

15.2.2.4 void DDS::rpc::example::Foo_Call::op2 (const struct Foo_op2_In & *_v*) [inline]

Set this when invoking the method 'op2'.

15.3 DDS::rpc::example::Foo_op1_In Struct Reference

Public Member Functions

- **Foo_op1_In** ()

Public Attributes

- int **param**

15.3.1 Detailed Description

Data type generated to capture the parameters to the interface method 'Foo::operation' Each parameter is mapped to a member of this data type.

15.3.2 Constructor & Destructor Documentation

15.3.2.1 DDS::rpc::example::Foo_op1_In::Foo_op1_In ( )

Constructor, Copy Constructor, Destructor, Assignment operator

15.3.3 Member Data Documentation

15.3.3.1 int DDS::rpc::example::Foo_op1_In::param

The parameter named 'param'.

15.4 DDS::rpc::example::Foo_op1_Out Struct Reference

Public Member Functions

- **Foo_op1_Out** ()

Public Attributes

- unsigned char **return_**

15.4.1 Detailed Description

Data type generated to capture the output parameters and return value of the '**Foo::op1** (p. 234)' method. Each inout or out parameter is mapped to a member of this data type. The return value is mapped to a member named 'return_'.

15.4.2 Constructor & Destructor Documentation

15.4.2.1 DDS::rpc::example::Foo_op1_Out::Foo_op1_Out ( )

Constructor, Copy Constructor, Destructor, Assignment operator

15.4.3 Member Data Documentation

15.4.3.1 unsigned char DDS::rpc::example::Foo_op1_Out::return_

The return value from the method.

15.5 DDS::rpc::example::Foo_op1_Result Class Reference

Public Member Functions

- DDS::rpc::UnknownException **unknownEx** () const
- void **unknownEx** (DDS::rpc::UnknownException _v)
- const struct **Foo_op1_Out** & **result** () const
- void **result** (const struct **Foo_op1_Out** &_v)

15.5.1 Detailed Description

Data type generated to capture the result of the method '**Foo::op1()** (p. 234)'. This data type is a union that encapsulates the **Foo_op1_Out** (p. 235) member and provides for the indication of an exception from the service.

15.5.2 Member Function Documentation

15.5.2.1 const struct Foo_op1_Out& DDS::rpc::example::Foo_op1_Result::result ( ) const [inline]

A successful invocation of 'op1 ' will set this member.

15.5.2.2 void DDS::rpc::example::Foo_op1_Result::result (const struct Foo_op1_Out & _v) [inline]

Helper to set this member.

15.5.2.3 DDS::rpc::UnknownException DDS::rpc::example::Foo_op1_Result::unknownEx ( ) const [inline]

If the service throws an exception, it will select this member.

15.5.2.4 void DDS::rpc::example::Foo_op1_Result::unknownEx (DDS::rpc::UnknownException _v) [inline]

Helper to set this member.

15.6 DDS::rpc::example::Foo_op2_In Struct Reference

Public Member Functions

- **Foo_op2_In** ()

Public Attributes

- DDS::rpc::UnusedMember **dummy**

15.6.1 Detailed Description

Data type generated to capture the parameters to the interface method 'Foo::operation' Each parameter is mapped to a member of this data type.

15.6.2 Constructor & Destructor Documentation

15.6.2.1 DDS::rpc::example::Foo_op2_In::Foo_op2_In ( )

Constructor, Copy Constructor, Destructor, Assignment operator

15.6.3 Member Data Documentation

15.6.3.1 DDS::rpc::UnusedMember DDS::rpc::example::Foo_op2_In::dummy

There are no parameters to this operation, so we add a dummy to avoid an empty structure

15.7 DDS::rpc::example::Foo_op2_Out Struct Reference

Public Member Functions

- **Foo_op2_Out** ()

Public Attributes

- int **return_**

15.7.1 Detailed Description

Data type generated to capture the result of the method '**Foo::op2()** (p. 234)'. This data type is a union that encapsulates the **Foo_op1_Out** (p. 235) member and provides for the indication of an exception from the service.

15.7.2 Constructor & Destructor Documentation

15.7.2.1 DDS::rpc::example::Foo_op2_Out::Foo_op2_Out ( )

Constructor, Copy Constructor, Destructor, Assignment operator

15.7.3 Member Data Documentation

15.7.3.1 int DDS::rpc::example::Foo_op2_Out::return_

The return value from the method.

15.8 DDS::rpc::example::Foo_op2_Result Class Reference

Public Member Functions

- DDS::rpc::UnknownException **unknownEx** () const
- void **unknownEx** (DDS::rpc::UnknownException _v)
- const struct **Foo_op2_Out** & **result** () const
- void **result** (const struct **Foo_op2_Out** &_v)

15.8.1 Detailed Description

Data type generated to capture the result of the method '**Foo::op2()** (p. 234)'. This data type is a union that encapsulates the **Foo_op2_Out** (p. 237) member and provides for the indication of an exception from the service.

15.8.2 Member Function Documentation

15.8.2.1 const struct Foo_op2_Out& DDS::rpc::example::Foo_op2_Result::result ( ) const [inline]

A successful invocation of 'op2 ' will set this member.

15.8.2.2 void DDS::rpc::example::Foo_op2_Result::result (const struct Foo_op2_Out & _v) [inline]

Helper to set this member.

15.8.2.3 DDS::rpc::UnknownException DDS::rpc::example::Foo_op2_Result::unknownEx ( ) const [inline]

If the service throws an exception, it will select this member.

15.8.2.4 void DDS::rpc::example::Foo_op2_Result::unknownEx (DDS::rpc::UnknownException _v) [inline]

Helper to set this member.

15.9 DDS::rpc::example::Foo_Reply Struct Reference

Public Member Functions

- **Foo_Reply** ()
- **DDS::SampleIdentity_t** ∗ **related_id** ()

Public Attributes

- struct DDS::rpc::ReplyHeader **header**
- class **Foo_Return data**

15.9.1 Detailed Description

Data type generated to represent a Reply from the **Foo** (p. 233) service.

The Reply data type includes the Return data type and an RPC header. The header indicates the 'relatedRequestId' that associates this Reply with a previous Request.

15.9.2 Constructor & Destructor Documentation

15.9.2.1 DDS::rpc::example::Foo_Reply::Foo_Reply ( )

Constructor, Copy Constructor, Destructor, Assignment operator

15.9.3 Member Function Documentation

15.9.3.1 DDS::SampleIdentity_t∗ DDS::rpc::example::Foo_Reply::related_id ( ) [inline]

Access the relatedRequestId from the RPC header

15.9.4 Member Data Documentation

15.9.4.1 class Foo_Return DDS::rpc::example::Foo_Reply::data

the **Foo_Return** (p. 240) union that holds the inout, out, and return values from an operation invocation.

15.9.4.2 struct DDS::rpc::ReplyHeader DDS::rpc::example::Foo_Reply::header

the RPC header, holding the relatedRequestId

15.10 DDS::rpc::example::Foo_Request Struct Reference

Public Member Functions

- **Foo_Request** ()
- **DDS::SampleIdentity_t** ∗ **request_id** ()

Public Attributes

- struct DDS::rpc::RequestHeader **header**
- class **Foo_Call data**

15.10.1 Detailed Description

Data type generated to represent a Request (call to one of the methods) on the **Foo** (p. 233) service.

15.10.2 Constructor & Destructor Documentation

15.10.2.1 DDS::rpc::example::Foo_Request::Foo_Request ( )

Constructor, Copy Constructor, Destructor, Assignment operator

15.10.3 Member Function Documentation

15.10.3.1 DDS::SampleIdentity_t* DDS::rpc::example::Foo_Request::request_id ( ) [inline]

Each request generated by a client has a unique requestId. This is populated when the request is published.

15.10.4 Member Data Documentation

15.10.4.1 class Foo_Call DDS::rpc::example::Foo_Request::data

This holds the parameter data to pass to the service and an indication of which method is being requested. (The union discriminator.)

15.10.4.2 struct DDS::rpc::RequestHeader DDS::rpc::example::Foo_Request::header

The Request includes a header that identifies the source GUID and SequenceNumber of the request.

15.11 DDS::rpc::example::Foo_Return Class Reference

Public Member Functions

- const class **Foo_op1_Result** & **op1** () const
- class **Foo_op1_Result** & **op1** ()
- const class **Foo_op2_Result** & **op2** () const
- void **op2** (const class **Foo_op2_Result** &_v)
- DDS::rpc::UnknownOperation **unknownOp** () const
- void **unknownOp** (DDS::rpc::UnknownOperation _v)

15.11.1 Detailed Description

Data type generated to represent the result from a method in the **Foo** (p. 233) service.

The Return data type is a union, with one case for each operation in the **Foo** (p. 233) service.

15.11.2 Member Function Documentation

15.11.2.1 const class Foo_op1_Result& DDS::rpc::example::Foo_Return::op1 ( ) const [inline]

Get the result of **op1()** (p. 241).

15.11.2.2 class Foo_op1_Result& DDS::rpc::example::Foo_Return::op1 ( ) [inline]

Set the result of **op1()** (p. 241).

15.11.2.3 const class Foo_op2_Result& DDS::rpc::example::Foo_Return::op2 ( ) const [inline]

Get the result of op2().

15.11.2.4 void DDS::rpc::example::Foo_Return::op2 (const class Foo_op2_Result & _*v*) [inline]

Set the result of op2().

15.11.2.5 DDS::rpc::UnknownOperation DDS::rpc::example::Foo_Return::unknownOp ( ) const [inline]

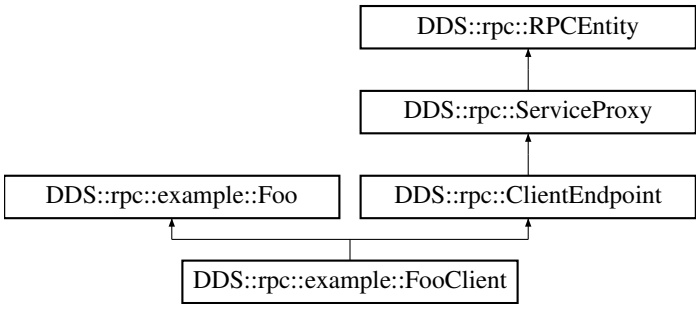
Get the UnknownOperation exception.

15.11.2.6 void DDS::rpc::example::Foo_Return::unknownOp (DDS::rpc::UnknownOperation _*v*) [inline]

Set the UnknownOperation exception.

15.12 DDS::rpc::example::FooClient Struct Reference

Inheritance diagram for DDS::rpc::example::FooClient:



Public Member Functions

- **FooClient** (**DDS::rpc::ClientParams** &params)
- unsigned char **op1** (const int param)
- int **op2** ()

15.12.1 Detailed Description

Data type generated to provide a Client interface to the **Foo** (p. 233) service.

This object derives from **Foo** (p. 233) and **ClientEndpoint** (p. 205).

Associated IDL:

```
@service
 interface Foo (p. 233) {
    boolean op1( long param );
    long    op2 (  ) (p. 242);
};
```

15.12.2 Constructor & Destructor Documentation

15.12.2.1 DDS::rpc::example::FooClient::FooClient (DDS::rpc::ClientParams & *params*)

Construct with **ClientParams** (p. 206) configuration

15.12.3 Member Function Documentation

15.12.3.1 unsigned char DDS::rpc::example::FooClient::op1 (const int *param*) [virtual]

Foo::op1() (p. 234) implementation that will publish the appropriate Request and handle the Reply.

Implements **DDS::rpc::example::Foo** (p. 234).

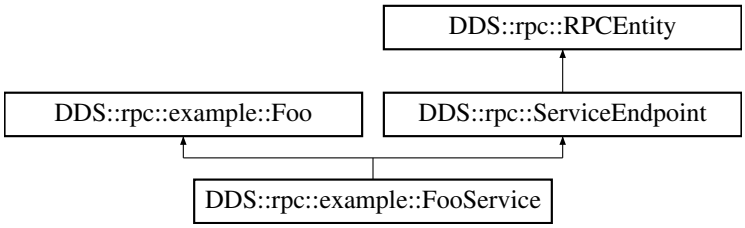
15.12.3.2 int DDS::rpc::example::FooClient::op2 ( ) [virtual]

Foo::op1() (p. 234) implementation that will publish the appropriate Request and handle the Reply.

Implements **DDS::rpc::example::Foo** (p. 234).

15.13 DDS::rpc::example::FooService Struct Reference

Inheritance diagram for DDS::rpc::example::FooService:



Public Member Functions

- **FooService** (**DDS::rpc::ServiceParams** &params)
- virtual unsigned char **op1** (const int param)=0
- virtual int **op2** ()=0

15.13.1 Detailed Description

Data type generated to provide a Client interface to the **Foo** (p. 233) service.

```
The FooService instantiates the underlying %DDS entities
and makes the service discoverable. Every Service inherits from
ServiceEndpoint and the service specific class, in this case, Foo.
A Service may belong to only one Server.
```

Associated IDL:

```
@service
 interface Foo (p. 233) {
    boolean op1( long param );
    long    op2(  ) (p. 243);
};
```

15.13.2 Constructor & Destructor Documentation

15.13.2.1 DDS::rpc::example::FooService::FooService (DDS::rpc::ServiceParams & *params*)

Construct with **ServiceParams** (p. 227) configuration

15.13.3 Member Function Documentation

15.13.3.1 virtual unsigned char DDS::rpc::example::FooService::op1 (const int *param*) [pure virtual]

Foo::op1() (p. 234) the application must derive from this class and provide an implementation of this method.

Implements **DDS::rpc::example::Foo** (p. 234).

15.13.3.2 virtual int DDS::rpc::example::FooService::op2 ( ) [pure virtual]

Foo::op1() (p. 234) the application must derive from this class and provide an implementation of this method.

Implements **DDS::rpc::example::Foo** (p. 234).

Chapter 16

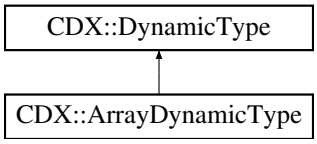
CoreDX DDS DynamicTypes

NOTE: This API is deprecated. The preferred DynamicType API is the one provided by the XTypes implementation.

16.1 CDX::ArrayDynamicType Class Reference

Contains an array.

Inheritance diagram for CDX::ArrayDynamicType:



Public Member Functions

- **DDS::ReturnCode_t set_element_type** (**DynamicType** ∗e)

  Defines the element type of an ARRAY or SEQUENCE.
- **DynamicType** ∗ **get_element_type** ()

  *Provides access to the type of the of data held in an ARRAY or SEQUENCE **DynamicType** (p. 251) object.*
- **DDS::ReturnCode_t set_element** (uint32_t n, **DynamicType** ∗e)

  Assigns a value to an element of an ARRAY or SEQUENCE.
- **DynamicType** ∗ **get_element** (uint32_t n)

  *Provides access to the a data element held in an ARRAY or SEQUENCE **DynamicType** (p. 251) object.*
- **DDS::ReturnCode_t set_max_length** (uint32_t n)

  *Assigns a 'max_length' value to the provided STRING, ARRAY, or SEQUENCE **DynamicType** (p. 251).*
- **DDS::ReturnCode_t set_length** (uint32_t n)

  *Assigns a 'length' value to the provided ARRAY or SEQUENCE **DynamicType** (p. 251).*
- uint32_t **get_length** ()

  *Provides access to the length of data held in a **DynamicType** (p. 251) object.*

16.1.1 Detailed Description

Contains an array.

16.1.2 Member Function Documentation

16.1.2.1 DynamicType* CDX::ArrayDynamicType::get_element (uint32_t *n*)

Provides access to the a data element held in an ARRAY or SEQUENCE **DynamicType** (p. 251) object.

This is applicable for a SEQUENCE or ARRAY **DynamicType** (p. 251) object. For a SEQUENCE, this returns the element 'n' of the sequence. For an ARRAY, this returns the element 'n' of the array elements. Elements are indexed starting at 0.

Return values

uint32_t	the data element held in the SEQUENCE or ARRAY 't' at index 'n'.

16.1.2.2 DynamicType* CDX::ArrayDynamicType::get_element_type ( )

Provides access to the type of the of data held in an ARRAY or SEQUENCE **DynamicType** (p. 251) object.

This is applicable for a SEQUENCE or ARRAY **DynamicType** (p. 251) object. For a SEQUENCE, this returns the 'type' of the sequence elements. For an ARRAY, this returns the 'type' of the array elements.

Return values

uint32_t	the 'type' of the data value held in the SEQUENCE or ARRAY 't'.

16.1.2.3 uint32_t CDX::ArrayDynamicType::get_length ( )

Provides access to the length of data held in a **DynamicType** (p. 251) object.

This is applicable for a SEQUENCE or ARRAY **DynamicType** (p. 251) object. For a SEQUENCE, this returns the 'length' of the sequence. For an ARRAY, this returns the size of the array.

Return values

uint32_t	the length of the data value held by 't'.

16.1.2.4 DDS::ReturnCode_t CDX::ArrayDynamicType::set_element (uint32_t *n*, DynamicType * *e*)

Assigns a value to an element of an ARRAY or SEQUENCE.

Note

The ARRAY or SEQUENCE takes ownership of the provided field.

Return values

BAD_PARAMETER	if 't' is not of type ARRAY or SEQUENCE or if 'n' is beyond the specified length of the ARRAY or SEQUENCE.
OK	upon success.

16.1.2.5 DDS::ReturnCode_t CDX::ArrayDynamicType::set_element_type (DynamicType * *e*)

Defines the element type of an ARRAY or SEQUENCE.

Return values

BAD_PARAMETER	if 't' is not of type ARRAY or SEQUENCE.
OK	upon success.

16.1.2.6 DDS::ReturnCode_t CDX::ArrayDynamicType::set_length (uint32_t *n*)

Assigns a 'length' value to the provided ARRAY or SEQUENCE **DynamicType** (p. 251).

This defines the actual size of the array or sequence data. This will allocate memory to hold 'n' entries. The entries are not initialized, and must be initialized by calling **set_element()** (p. 246) 'n' times.

Note

If **set_length()** (p. 247) has been called previously on object 't', the subsequent calls to this routine will deallocate the storate for the previous array or sequence elements. However, the elements themselves will not be freed. To avoid a memory leak, it is necessary to manually free each element before calling **set_length()** (p. 247).

Return values

BAD_PARAMETER	if 't' is not of type ARRAY or SEQUENCE; OUT_OF_RESOURCES if memory allocation fails.
OK	upon success.

16.1.2.7 DDS::ReturnCode_t CDX::ArrayDynamicType::set_max_length (uint32_t *n*)

Assigns a 'max_length' value to the provided STRING, ARRAY, or SEQUENCE **DynamicType** (p. 251).

For STRINGS and SEQUENCES, this defines the bound on the string length or sequence length. For arrays, this defines the size of the array.

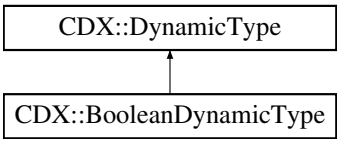
Return values

BAD_PARAMETER	if 't' is not of type STRING, ARRAY, or SEQUENCE.
OK	upon success.

16.2 CDX::BooleanDynamicType Class Reference

Contains a boolean.

Inheritance diagram for CDX::BooleanDynamicType:



Public Member Functions

- unsigned char **get_boolean** ()

  *Provides access to data held in an BOOLEAN **DynamicType** (p. 251) object.*
- **DDS::ReturnCode_t set_boolean** (unsigned char c)

  *Assigns a value to the provided BOOLEAN **DynamicType** (p. 251).*

16.2.1 Detailed Description

Contains a boolean.

16.2.2 Member Function Documentation

16.2.2.1 unsigned char CDX::BooleanDynamicType::get_boolean ( )

Provides access to data held in an BOOLEAN **DynamicType** (p. 251) object.

Return values

unsigned_char	0 indicates FALSE, non-zero indicates TRUE.

16.2.2.2 DDS::ReturnCode_t CDX::BooleanDynamicType::set_boolean (unsigned char *c*)

Assigns a value to the provided BOOLEAN **DynamicType** (p. 251).

Return values

BAD_PARAMETER	if 't' is not of type BOOLEAN.
OK	upon success.

16.3 CDX_XmlApi Struct Reference

Provides a generic XML Parser capability.

16.3.1 Detailed Description

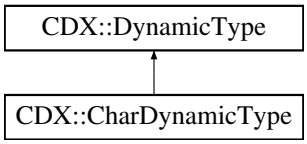
Provides a generic XML Parser capability.

An implementation of this API is used by CoreDX **DDS** (p. 313) to parse XML documents. This API hides the details of the XML Parser technology, allowing the user to provide their own XML Parser as required. CoreDX DDS provides an implementation based on libxml2 as a default. There is also an implementation based on RapidXML available.

16.4 CDX::CharDynamicType Class Reference

Contains a char.

Inheritance diagram for CDX::CharDynamicType:



Public Member Functions

- char **get_char** ()

  *Provides access to data held in an CHAR **DynamicType** (p. 251) object.*
- **DDS::ReturnCode_t set_char** (char c)

  *Assigns a value to the provided CHAR **DynamicType** (p. 251).*

16.4.1 Detailed Description

Contains a char.

16.4.2 Member Function Documentation

16.4.2.1 char CDX::CharDynamicType::get_char ( )

Provides access to data held in an CHAR **DynamicType** (p. 251) object.

Return values

char	the data value held by 't'.

16.4.2.2 DDS::ReturnCode_t CDX::CharDynamicType::set_char (char *c*)

Assigns a value to the provided CHAR **DynamicType** (p. 251).

Return values

BAD_PARAMETER	if 't' is not of type CHAR.

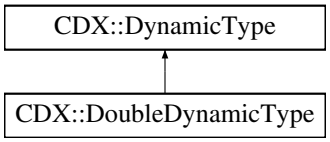
Return values

OK	upon success.

16.5 CDX::DoubleDynamicType Class Reference

Contains a double.

Inheritance diagram for CDX::DoubleDynamicType:



Public Member Functions

- double **get_double** ()

  *Provides access to data held in an DOUBLE **DynamicType** (p. 251) object.*
- **DDS::ReturnCode_t set_double** (double c)

  *Assigns a value to the provided DOUBLE **DynamicType** (p. 251).*

16.5.1 Detailed Description

Contains a double.

16.5.2 Member Function Documentation

16.5.2.1 double CDX::DoubleDynamicType::get_double ( )

Provides access to data held in an DOUBLE **DynamicType** (p. 251) object.

Return values

double	the data value held by 't'.

16.5.2.2 DDS::ReturnCode_t CDX::DoubleDynamicType::set_double (double *c*)

Assigns a value to the provided DOUBLE **DynamicType** (p. 251).

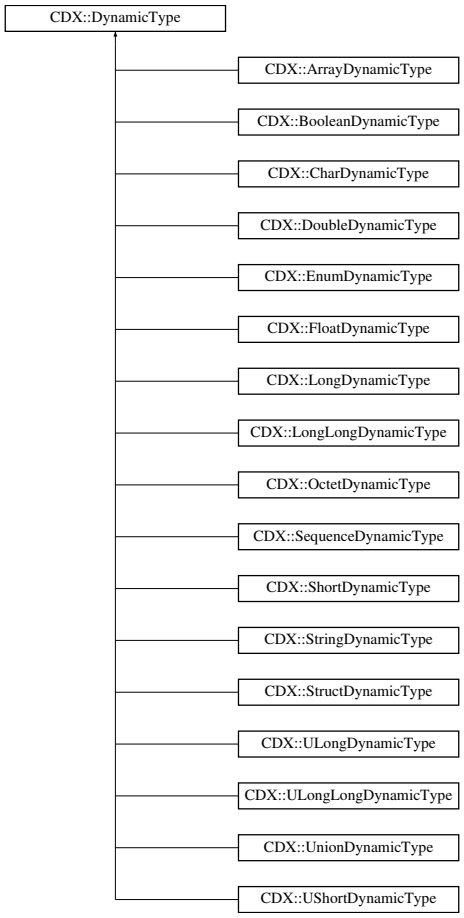
Return values

BAD_PARAMETER	if 't' is not of type DOUBLE.
OK	upon success.

16.6 CDX::DynamicType Class Reference

DynamicType (p. 251) is an object that enhances CoreDX DDS with the facilities to process dynamic data types (in other words, data types that are not defined at compile time).

Inheritance diagram for CDX::DynamicType:



Public Member Functions

- virtual **TypeCodeKind get_type** ()

  *Provides access to the 'type' of the **DynamicType** (p. 251) object. Applicable to any **DynamicType** (p. 251).*

16.6.1 Detailed Description

DynamicType (p. 251) is an object that enhances CoreDX DDS with the facilities to process dynamic data types (in other words, data types that are not defined at compile time).

The ability to process dynamic types adds significant flexibility to CoreDX DDS. The type information can be discovered at run-time, constructed programmatically, or based on existing type information. The **DynamicType** (p. 251) support allows the application to process a data type without compiling and linking type specific source code. For an application that must support a large number of data types, this can offer a savings in code size.

16.6.2 Member Function Documentation

16.6.2.1 virtual TypeCodeKind CDX::DynamicType::get_type ( ) [virtual]

Provides access to the 'type' of the **DynamicType** (p. 251) object. Applicable to any **DynamicType** (p. 251).

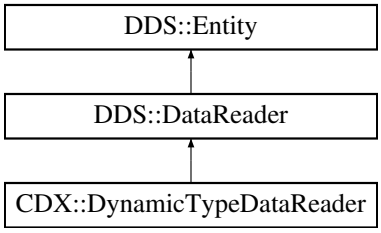
Return values

TypeCodeKind	the type of object 't'.

16.7 CDX::DynamicTypeDataReader Class Reference

A DataReader that reads '**DynamicType** (p. 251)' data elements A **DynamicTypeDataReader** (p. 252) can read data published by a normal DataWriter or by a **DynamicTypeDataWriter** (p. 252), if the QoS and data types match.

Inheritance diagram for CDX::DynamicTypeDataReader:



Additional Inherited Members

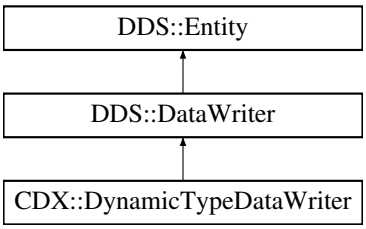
16.7.1 Detailed Description

A DataReader that reads '**DynamicType** (p. 251)' data elements A **DynamicTypeDataReader** (p. 252) can read data published by a normal DataWriter or by a **DynamicTypeDataWriter** (p. 252), if the QoS and data types match.

16.8 CDX::DynamicTypeDataWriter Class Reference

A DataWriter that writes '**DynamicType** (p. 251)' data elements. Data written by this DataWriter can be read by a standard DataReader or by a **DynamicTypeDataReader** (p. 252), if the QoS and data types match.

Inheritance diagram for CDX::DynamicTypeDataWriter:



Additional Inherited Members

16.8.1 Detailed Description

A DataWriter that writes '**DynamicType** (p. 251)' data elements. Data written by this DataWriter can be read by a standard DataReader or by a **DynamicTypeDataReader** (p. 252), if the QoS and data types match.

16.9 CDX::EnumConstant Class Reference

Represents an enum constant value.

Public Attributes

- char ∗ **name**
- uint32_t **value**

16.9.1 Detailed Description

Represents an enum constant value.

16.9.2 Member Data Documentation

16.9.2.1 char∗ CDX::EnumConstant::name

The nul termintaed string name of this enum constatnt

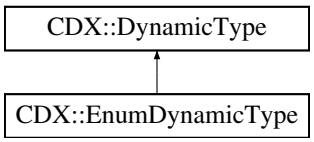
16.9.2.2 uint32_t CDX::EnumConstant::value

The integer value associated with this enum constant

16.10 CDX::EnumDynamicType Class Reference

Represents an enum (set of constants)

Inheritance diagram for CDX::EnumDynamicType:



Public Member Functions

- **DDS::ReturnCode_t set_num_constants** (int32_t n)
- **DDS::ReturnCode_t set_constant** (int32_t n, const char ∗name, uint32_t val)

- **DDS::ReturnCode_t set_value** (uint32_t c)
- int32_t **get_num_constants** (void)
- **EnumConstant** ∗ **get_constant** (int32_t index)
- **EnumConstant** ∗ **get_constant_by_name** (const char ∗name)
- **EnumConstant** ∗ **get_constant_by_value** (uint32_t val)
- uint32_t **get_value** (void)

16.10.1 Detailed Description

Represents an enum (set of constants)

16.10.2 Member Function Documentation

16.10.2.1 EnumConstant∗ CDX::EnumDynamicType::get_constant (int32_t *index*)

Get the name and value of a specific enumerator constant defined in this collection

16.10.2.2 EnumConstant∗ CDX::EnumDynamicType::get_constant_by_name (const char ∗ *name*)

Get the name and value of a specific enumerator by name

16.10.2.3 EnumConstant∗ CDX::EnumDynamicType::get_constant_by_value (uint32_t *val*)

Get the name and value of a specific enumerator by value

16.10.2.4 int32_t CDX::EnumDynamicType::get_num_constants (void)

Get the number of constants defined in this enumerator collection

16.10.2.5 uint32_t CDX::EnumDynamicType::get_value (void)

Get the value of an instance of the enumerator collection

16.10.2.6 DDS::ReturnCode_t CDX::EnumDynamicType::set_constant (int32_t *n*, const char ∗ *name*, uint32_t *val*)

Set the 'name' and 'val' of a specific enum constant

16.10.2.7 DDS::ReturnCode_t CDX::EnumDynamicType::set_num_constants (int32_t *n*)

Set the number of constants in the enumerator collection

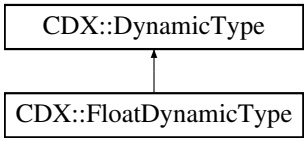
16.10.2.8 DDS::ReturnCode_t CDX::EnumDynamicType::set_value (uint32_t *c*)

Set the value of an instance of the enumerator collection

16.11 CDX::FloatDynamicType Class Reference

Contains a float.

Inheritance diagram for CDX::FloatDynamicType:



Public Member Functions

- float **get_float** ()

  *Provides access to data held in an FLOAT **DynamicType** (p. 251) object.*
- **DDS::ReturnCode_t set_float** (float c)

  *Assigns a value to the provided FLOAT **DynamicType** (p. 251).*

16.11.1 Detailed Description

Contains a float.

16.11.2 Member Function Documentation

16.11.2.1 float CDX::FloatDynamicType::get_float ( )

Provides access to data held in an FLOAT **DynamicType** (p. 251) object.

Return values

float	the data value held by 't'.

16.11.2.2 DDS::ReturnCode_t CDX::FloatDynamicType::set_float (float *c*)

Assigns a value to the provided FLOAT **DynamicType** (p. 251).

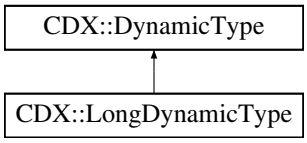
Return values

BAD_PARAMETER	if 't' is not of type FLOAT.
OK	upon success.

16.12 CDX::LongDynamicType Class Reference

Contains a long.

Inheritance diagram for CDX::LongDynamicType:



Public Member Functions

- int32_t **get_long** ()

  *Provides access to data held in an LONG **DynamicType** (p. 251) object.*
- **DDS::ReturnCode_t set_long** (int32_t c)

  *Assigns a value to the provided LONG **DynamicType** (p. 251).*

16.12.1 Detailed Description

Contains a long.

16.12.2 Member Function Documentation

16.12.2.1 int32_t CDX::LongDynamicType::get_long ( )

Provides access to data held in an LONG **DynamicType** (p. 251) object.

Return values

int32_t	the data value held by 't'.

16.12.2.2 DDS::ReturnCode_t CDX::LongDynamicType::set_long (int32_t *c*)

Assigns a value to the provided LONG **DynamicType** (p. 251).

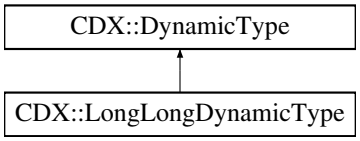
Return values

BAD_PARAMETER	if 't' is not of type LONG.
OK	upon success.

16.13 CDX::LongLongDynamicType Class Reference

Contains a long long.

Inheritance diagram for CDX::LongLongDynamicType:



Public Member Functions

- int64_t **get_longlong** ()

  *Provides access to data held in an LONG LONG **DynamicType** (p. 251) object.*
- **DDS::ReturnCode_t set_longlong** (int64_t c)

  *Assigns a value to the provided LONGLONG **DynamicType** (p. 251).*

16.13.1 Detailed Description

Contains a long long.

16.13.2 Member Function Documentation

16.13.2.1 int64_t CDX::LongLongDynamicType::get_longlong ( )

Provides access to data held in an LONG LONG **DynamicType** (p. 251) object.

Return values

int64_t	the data value held by 't'.

16.13.2.2 DDS::ReturnCode_t CDX::LongLongDynamicType::set_longlong (int64_t *c*)

Assigns a value to the provided LONGLONG **DynamicType** (p. 251).

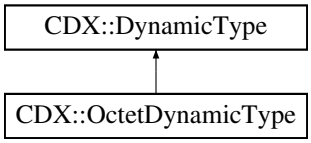
Return values

BAD_PARAMETER	if 't' is not of type LONGLONG.
OK	upon success.

16.14 CDX::OctetDynamicType Class Reference

Contains an octet.

Inheritance diagram for CDX::OctetDynamicType:



Public Member Functions

- unsigned char **get_octet** ()

  *Provides access to data held in an OCTET **DynamicType** (p. 251) object.*
- **DDS::ReturnCode_t set_octet** (unsigned char c)

  *Assigns a value to the provided OCTET **DynamicType** (p. 251).*

16.14.1 Detailed Description

Contains an octet.

16.14.2 Member Function Documentation

16.14.2.1 unsigned char CDX::OctetDynamicType::get_octet ( )

Provides access to data held in an OCTET **DynamicType** (p. 251) object.

Return values

unsigned_char	the data value held by 't'.

16.14.2.2 DDS::ReturnCode_t CDX::OctetDynamicType::set_octet (unsigned char *c*)

Assigns a value to the provided OCTET **DynamicType** (p. 251).

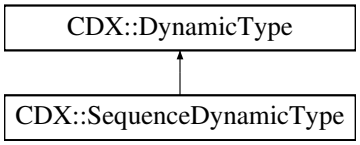
Return values

BAD_PARAMETER	if 't' is not of type OCTET.
OK	upon success.

16.15 CDX::SequenceDynamicType Class Reference

Contains a sequence.

Inheritance diagram for CDX::SequenceDynamicType:



Public Member Functions

- uint32_t **get_length** ()

  *Provides access to the length of data held in a **DynamicType** (p. 251) object.*
- **DynamicType** ∗ **get_element_type** ()

  *Provides access to the type of the of data held in an ARRAY or SEQUENCE **DynamicType** (p. 251) object.*
- **DynamicType** ∗ **get_element** (uint32_t n)

  *Provides access to the a data element held in an ARRAY or SEQUENCE **DynamicType** (p. 251) object.*
- **DDS::ReturnCode_t set_max_length** (uint32_t n)

  *Assigns a 'max_length' value to the provided STRING, ARRAY, or SEQUENCE **DynamicType** (p. 251).*
- **DDS::ReturnCode_t set_length** (uint32_t n)

  *Assigns a 'length' value to the provided ARRAY or SEQUENCE **DynamicType** (p. 251).*
- **DDS::ReturnCode_t set_element_type** (**DynamicType** ∗e)

  Defines the element type of an ARRAY or SEQUENCE.
- **DDS::ReturnCode_t set_element** (uint32_t n, **DynamicType** ∗e)

  Assigns a value to an element of an ARRAY or SEQUENCE.

16.15.1 Detailed Description

Contains a sequence.

16.15.2 Member Function Documentation

16.15.2.1 DynamicType∗ CDX::SequenceDynamicType::get_element (uint32_t *n*)

Provides access to the a data element held in an ARRAY or SEQUENCE **DynamicType** (p. 251) object.

This is applicable for a SEQUENCE or ARRAY **DynamicType** (p. 251) object. For a SEQUENCE, this returns the element 'n' of the sequence. For an ARRAY, this returns the element 'n' of the array elements. Elements are indexed starting at 0.

Return values

uint32_t	the data element held in the SEQUENCE or ARRAY 't' at index 'n'.

16.15.2.2 DynamicType∗ CDX::SequenceDynamicType::get_element_type ( )

Provides access to the type of the of data held in an ARRAY or SEQUENCE **DynamicType** (p. 251) object.

This is applicable for a SEQUENCE or ARRAY **DynamicType** (p. 251) object. For a SEQUENCE, this returns the 'type' of the sequence elements. For an ARRAY, this returns the 'type' of the array elements.

Return values

uint32_t	the 'type' of the data value held in the SEQUENCE or ARRAY 't'.

16.15.2.3 uint32_t CDX::SequenceDynamicType::get_length ( )

Provides access to the length of data held in a **DynamicType** (p. 251) object.

This is applicable for a SEQUENCE or ARRAY **DynamicType** (p. 251) object. For a SEQUENCE, this returns the 'length' of the sequence. For an ARRAY, this returns the size of the array.

Return values

uint32_t	the length of the data value held by 't'.

16.15.2.4 DDS::ReturnCode_t CDX::SequenceDynamicType::set_element (uint32_t *n*, DynamicType * *e*)

Assigns a value to an element of an ARRAY or SEQUENCE.

Note

The ARRAY or SEQUENCE takes ownership of the provided field.

Return values

BAD_PARAMETER	if 't' is not of type ARRAY or SEQUENCE or if 'n' is beyond the specified length of the ARRAY or SEQUENCE.
OK	upon success.

16.15.2.5 DDS::ReturnCode_t CDX::SequenceDynamicType::set_element_type (DynamicType * *e*)

Defines the element type of an ARRAY or SEQUENCE.

Return values

BAD_PARAMETER	if 't' is not of type ARRAY or SEQUENCE.
OK	upon success.

16.15.2.6 DDS::ReturnCode_t CDX::SequenceDynamicType::set_length (uint32_t *n*)

Assigns a 'length' value to the provided ARRAY or SEQUENCE **DynamicType** (p. 251).

This defines the actual size of the array or sequence data. This will allocate memory to hold 'n' entries. The entries are not initialized, and must be initialized by calling **set_element()** (p. 260) 'n' times.

Note

If **set_length()** (p. 260) has been called previously on object 't', the subsequent calls to this routine will deallocate the storate for the previous array or sequence elements. However, the elements themselves will not be freed. To avoid a memory leak, it is necessary to manually free each element before calling **set_length()** (p. 260).

Return values

BAD_PARAMETER	if 't' is not of type ARRAY or SEQUENCE; OUT_OF_RESOURCES if memory allocation fails.
OK	upon success.

16.15.2.7 DDS::ReturnCode_t CDX::SequenceDynamicType::set_max_length (uint32_t *n*)

Assigns a 'max_length' value to the provided STRING, ARRAY, or SEQUENCE **DynamicType** (p. 251).

For STRINGS and SEQUENCES, this defines the bound on the string length or sequence length. For arrays, this defines the size of the array.

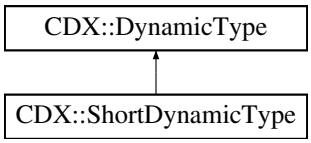
Return values

BAD_PARAMETER	if 't' is not of type STRING, ARRAY, or SEQUENCE.
OK	upon success.

16.16 CDX::ShortDynamicType Class Reference

Contains a short.

Inheritance diagram for CDX::ShortDynamicType:



Public Member Functions

- int16_t **get_short** ()

  *Provides access to data held in an SHORT **DynamicType** (p. 251) object.*
- **DDS::ReturnCode_t set_short** (short c)

  *Assigns a value to the provided SHORT **DynamicType** (p. 251).*

16.16.1 Detailed Description

Contains a short.

16.16.2 Member Function Documentation

16.16.2.1 int16_t CDX::ShortDynamicType::get_short ( )

Provides access to data held in an SHORT **DynamicType** (p. 251) object.

Return values

int16_t	the data value held by 't'.

16.16.2.2 DDS::ReturnCode_t CDX::ShortDynamicType::set_short (short *c*)

Assigns a value to the provided SHORT **DynamicType** (p. 251).

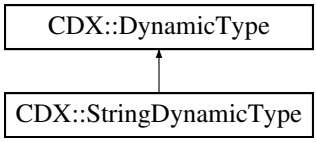
Return values

BAD_PARAMETER	if 't' is not of type SHORT.
OK	upon success.

16.17 CDX::StringDynamicType Class Reference

Contains a string.

Inheritance diagram for CDX::StringDynamicType:



Public Member Functions

- const char ∗ **get_string** ()

  *Provides access to data held in an STRING **DynamicType** (p. 251) object.*
- uint32_t **get_max_length** ()

  *Provides access to the maximum length of a **DynamicType** (p. 251) object.*
- uint32_t **get_length** ()

  *Provides access to the length of data held in a **DynamicType** (p. 251) object.*
- **DDS::ReturnCode_t set_string** (const char ∗c)

  *Assigns a value to the provided STRING **DynamicType** (p. 251).*
- **DDS::ReturnCode_t set_max_length** (uint32_t n)

  *Assigns a 'max_length' value to the provided STRING, ARRAY, or SEQUENCE **DynamicType** (p. 251).*

16.17.1 Detailed Description

Contains a string.

16.17.2 Member Function Documentation

16.17.2.1 uint32_t CDX::StringDynamicType::get_length ( )

Provides access to the length of data held in a **DynamicType** (p. 251) object.

This is applicable for a SEQUENCE or ARRAY **DynamicType** (p. 251) object. For a SEQUENCE, this returns the 'length' of the sequence. For an ARRAY, this returns the size of the array.

Return values

uint32_t	the length of the data value held by 't'.

16.17.2.2 uint32_t CDX::StringDynamicType::get_max_length ( )

Provides access to the maximum length of a **DynamicType** (p. 251) object.

This is applicable for a STRING, SEQUENCE, or ARRAY **DynamicType** (p. 251) object. For a STRING, this returns the 'fixed length' of the string, or zero if the string is not fixed length. For a SEQUENCE, this returns the 'fixed length' of the sequence, or zero if the sequence is unbounded. For an ARRAY, this returns the size of the array.

Return values

uint32_t	the maximum length of the data value held by 't'.

16.17.2.3 const char* CDX::StringDynamicType::get_string ( )

Provides access to data held in an STRING **DynamicType** (p. 251) object.

Return values

*const_char_**	the data value held by 't'.

16.17.2.4 DDS::ReturnCode_t CDX::StringDynamicType::set_max_length (uint32_t *n*)

Assigns a 'max_length' value to the provided STRING, ARRAY, or SEQUENCE **DynamicType** (p. 251).

For STRINGS and SEQUENCES, this defines the bound on the string length or sequence length. For arrays, this defines the size of the array.

Return values

BAD_PARAMETER	if 't' is not of type STRING, ARRAY, or SEQUENCE.
OK	upon success.

16.17.2.5 DDS::ReturnCode_t CDX::StringDynamicType::set_string (const char * *c*)

Assigns a value to the provided STRING **DynamicType** (p. 251).

This routine will make a copy of the provided string data. If the STRING has a defined 'max_length', then the copied data will be truncated to 'max_length' characters. [A '\0' (nul) character will be added after the truncated data.]

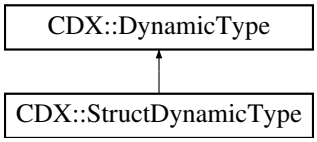
Return values

DDS::ReturnCode_t *(p. 318)*	BAD_PARAMETER if 't' is not of type STRING.
OUT_OF_RESOURCES	if memory allocation fails.
OK	upon success.

16.18 CDX::StructDynamicType Class Reference

Represents a struct of members. Each member has an associated data type.

Inheritance diagram for CDX::StructDynamicType:



Public Member Functions

- uint32_t **get_num_fields** ()

  *Provides access to the number of fields held by a STRUCT or UNION **DynamicType** (p. 251) object.*
- **DynamicType** * **get_field** (uint32_t n)

  *Provides access to a field held by a STRUCT or UNION **DynamicType** (p. 251) object.*
- const char * **get_field_name** (uint32_t n)

  *Provides access to the name of a field held by a STRUCT or UNION **DynamicType** (p. 251) object.*
- unsigned char **get_field_key** (uint32_t n)

  *Provides access to the 'key' indication for a field held by a STRUCT **DynamicType** (p. 251) object.*
- **DDS::ReturnCode_t set_num_fields** (uint32_t n)

  *Defines the number of fields held by a STRUCT or UNION **DynamicType** (p. 251) object.*
- **DDS::ReturnCode_t set_field** (uint32_t n, const char *field_name, **DynamicType** *e, unsigned char key)

  *Assigns a value to a field of a STRUCT or UNION **DynamicType** (p. 251) object.*

16.18.1 Detailed Description

Represents a struct of members. Each member has an associated data type.

16.18.2 Member Function Documentation

16.18.2.1 DynamicType* CDX::StructDynamicType::get_field (uint32_t *n*)

Provides access to a field held by a STRUCT or UNION **DynamicType** (p. 251) object.

This is applicable for a STRUCT or UNION **DynamicType** (p. 251) object. For a STRUCT or UNION, this returns a field held in the data structure.

Return values

DynamicType *(p. 251)*	the number of data fields in the STRUCT or UNION 't'.

16.18.2.2 unsigned char CDX::StructDynamicType::get_field_key (uint32_t *n*)

Provides access to the 'key' indication for a field held by a STRUCT **DynamicType** (p. 251) object.

This is applicable for a STRUCT **DynamicType** (p. 251) object. For a STRUCT, this returns an indication that field 'n' is (or is not) a key in the data structure.

Return values

unsigned_char	if non-zero, the field at index 'n' is a key field. if zero, the field at index 'n' is not a key field.

16.18.2.3 const char* CDX::StructDynamicType::get_field_name (uint32_t *n*)

Provides access to the name of a field held by a STRUCT or UNION **DynamicType** (p. 251) object.

This is applicable for a STRUCT or UNION **DynamicType** (p. 251) object. For a STRUCT or UNION, this returns the name of a field held in the data structure.

Return values

*const_char_**	field name of field 'n' in STRUCT or UNION 't'.

16.18.2.4 uint32_t CDX::StructDynamicType::get_num_fields ( )

Provides access to the number of fields held by a STRUCT or UNION **DynamicType** (p. 251) object.

This is applicable for a STRUCT or UNION **DynamicType** (p. 251) object. For a STRUCT, this returns the number of fields held in the structure. For a UNION, this returns the number of fields contained by the UNION where a field is selectable by one or more case labels.

Return values

uint32_t	the number of data fields in the STRUCT or UNION 't'.

16.18.2.5 DDS::ReturnCode_t CDX::StructDynamicType::set_field (uint32_t *n*, const char * *field_name*, DynamicType * *e*, unsigned char *key*)

Assigns a value to a field of a STRUCT or UNION **DynamicType** (p. 251) object.

The 'nth' field of the struct or union is assigned the provided **field_name**, type **e**. The **key** parameter is used only for STRUCT types. If key is non-zero, then the field is added to the 'key set' for this data type. This routine makes a copy of the 'field_name' argument.

Note

The STRUCT or UNION takes ownership of the provided field.
Fields of a UNION data type can not be part of a 'key set', as the field may not exist in a particular instantiation of the union data type.

Return values

BAD_PARAMETER	if 't' is not of type STRUCT or UNION.
OUT_OF_MEMORY	if memory allocation (to hold a copy of field_name) fails.
OK	upon success.

16.18.2.6 DDS::ReturnCode_t CDX::StructDynamicType::set_num_fields (uint32_t *n*)

Defines the number of fields held by a STRUCT or UNION **DynamicType** (p. 251) object.

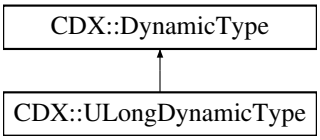
Return values

BAD_PARAMETER	if 't' is not of type STRUCT or UNION.
OK	upon success.

16.19 CDX::ULongDynamicType Class Reference

Contains an unsigned long.

Inheritance diagram for CDX::ULongDynamicType:



Public Member Functions

- uint32_t **get_ulong** ()

  *Provides access to data held in an UNSIGNED LONG **DynamicType** (p. 251) object.*
- **DDS::ReturnCode_t set_ulong** (uint32_t c)

*Assigns a value to the provided ULONG **DynamicType** (p. 251).*

16.19.1 Detailed Description

Contains an unsigned long.

16.19.2 Member Function Documentation

16.19.2.1 uint32_t CDX::ULongDynamicType::get_ulong ( )

Provides access to data held in an UNSIGNED LONG **DynamicType** (p. 251) object.

Return values

uint32_t	the data value held by 't'.

16.19.2.2 DDS::ReturnCode_t CDX::ULongDynamicType::set_ulong (uint32_t *c*)

Assigns a value to the provided ULONG **DynamicType** (p. 251).

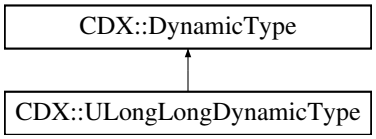
Return values

BAD_PARAMETER	if 't' is not of type ULONG.
OK	upon success.

16.20 CDX::ULongLongDynamicType Class Reference

Contains an unsigned long long.

Inheritance diagram for CDX::ULongLongDynamicType:



Public Member Functions

- uint64_t **get_ulonglong** ()

  *Provides access to data held in an UNSIGNED LONG LONG **DynamicType** (p. 251) object.*
- **DDS::ReturnCode_t set_ulonglong** (uint64_t c)

  *Assigns a value to the provided ULONGLONG **DynamicType** (p. 251).*

16.20.1 Detailed Description

Contains an unsigned long long.

16.20.2 Member Function Documentation

16.20.2.1 uint64_t CDX::ULongLongDynamicType::get_ulonglong ( )

Provides access to data held in an UNSIGNED LONG LONG **DynamicType** (p. 251) object.

Return values

uint64_t	the data value held by 't'.

16.20.2.2 DDS::ReturnCode_t CDX::ULongLongDynamicType::set_ulonglong (uint64_t *c*)

Assigns a value to the provided ULONGLONG **DynamicType** (p. 251).

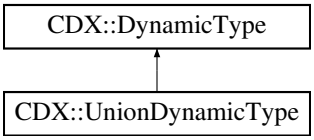
Return values

BAD_PARAMETER	if 't' is not of type ULONGLONG.
OK	upon success.

16.21 CDX::UnionDynamicType Class Reference

Represents a union.

Inheritance diagram for CDX::UnionDynamicType:



Public Member Functions

- uint32_t **get_num_fields** ()

  *Provides access to the number of fields held by a STRUCT or UNION **DynamicType** (p. 251) object.*
- **DynamicType** ∗ **get_field** (uint32_t n)

  *Provides access to a field held by a STRUCT or UNION **DynamicType** (p. 251) object.*
- const char ∗ **get_field_name** (uint32_t n)

  *Provides access to the name of a field held by a STRUCT or UNION **DynamicType** (p. 251) object.*
- **DynamicType** ∗ **get_discriminator** ()

  *Provides access to the 'discriminator' type of a UNION **DynamicType** (p. 251) object.*

- int32_t **get_default_field** ()

  *Provides access to the 'default' field of a UNION **DynamicType** (p. 251) object.*
- uint32_t **get_field_num_labels** (uint32_t field)

  *Provides access to the number of labels assigned to a field in the UNION **DynamicType** (p. 251) object.*
- int32_t **get_field_label** (uint32_t field, uint32_t label)

  *Provides access to the number of labels assigned to a field in the UNION **DynamicType** (p. 251) object.*
- **DynamicType** ∗ **get_selected_field** (void)

  Provides access to the field selected by the discriminator (or the default).
- **DDS::ReturnCode_t set_num_fields** (uint32_t n)

  *Defines the number of fields held by a STRUCT or UNION **DynamicType** (p. 251) object.*
- **DDS::ReturnCode_t set_field** (uint32_t n, const char ∗field_name, **DynamicType** ∗e)

  *Assigns a value to a field of a STRUCT or UNION **DynamicType** (p. 251) object.*
- **DDS::ReturnCode_t set_discriminator** (**DynamicType** ∗d)

  Defines the type and value of the UNION discriminator.
- **DDS::ReturnCode_t set_default_field** (int32_t field)

  Defines the index of the default field within the UNION.
- **DDS::ReturnCode_t set_field_num_labels** (uint32_t field, uint32_t n)

  Defines the number of labels associated with a field within the UNION.
- **DDS::ReturnCode_t set_field_label** (uint32_t field, uint32_t label, int32_t val)

  Assigns a label to the specified field within the UNION.

16.21.1 Detailed Description

Represents a union.

16.21.2 Member Function Documentation

16.21.2.1 int32_t CDX::UnionDynamicType::get_default_field ( )

Provides access to the 'default' field of a UNION **DynamicType** (p. 251) object.

This is applicable only for a UNION **DynamicType** (p. 251) object. This returns the index of the 'default' field in the UNION, if there is no field marked with a 'default:' case label.

Return values

int32_t	the index of the 'default' field. -1 if there is no defined 'default' case.

16.21.2.2 DynamicType∗ CDX::UnionDynamicType::get_discriminator ( )

Provides access to the 'discriminator' type of a UNION **DynamicType** (p. 251) object.

This is applicable only for a UNION **DynamicType** (p. 251) object. This returns the **DynamicType** (p. 251) that serves as a discriminator for the UNION data structure.

Return values

DynamicType *(p. 251)*	the discriminator data.

16.21.2.3 DynamicType* CDX::UnionDynamicType::get_field (uint32_t *n*)

Provides access to a field held by a STRUCT or UNION **DynamicType** (p. 251) object.

This is applicable for a STRUCT or UNION **DynamicType** (p. 251) object. For a STRUCT or UNION, this returns a field held in the data structure.

Return values

DynamicType *(p. 251)*	the number of data fields in the STRUCT or UNION 't'.

16.21.2.4 int32_t CDX::UnionDynamicType::get_field_label (uint32_t *field,* uint32_t *label*)

Provides access to the number of labels assigned to a field in the UNION **DynamicType** (p. 251) object.

This is applicable only for a UNION **DynamicType** (p. 251) object. This returns the count of the number of labels used to select a particular field in the UNION. The 'default:' label is not included in this count - as a result, it is possible for one field to have 'zero' labels.

Return values

int32_t	the number of labels assigned to field identified by index 'field'.

16.21.2.5 const char* CDX::UnionDynamicType::get_field_name (uint32_t *n*)

Provides access to the name of a field held by a STRUCT or UNION **DynamicType** (p. 251) object.

This is applicable for a STRUCT or UNION **DynamicType** (p. 251) object. For a STRUCT or UNION, this returns the name of a field held in the data structure.

Return values

*const_char_**	field name of field 'n' in STRUCT or UNION 't'.

16.21.2.6 uint32_t CDX::UnionDynamicType::get_field_num_labels (uint32_t *field*)

Provides access to the number of labels assigned to a field in the UNION **DynamicType** (p. 251) object.

This is applicable only for a UNION **DynamicType** (p. 251) object. This returns the count of the number of labels used to select a particular field in the UNION. The 'default:' label is not included in this count.

Return values

int32_t	the number of labels assigned to field identified by index 'field'.

16.21.2.7 uint32_t CDX::UnionDynamicType::get_num_fields ( )

Provides access to the number of fields held by a STRUCT or UNION **DynamicType** (p. 251) object.

This is applicable for a STRUCT or UNION **DynamicType** (p. 251) object. For a STRUCT, this returns the number of fields held in the structure. For a UNION, this returns the number of fields contained by the UNION where a field is selectable by one or more case labels.

Return values

uint32_t	the number of data fields in the STRUCT or UNION 't'.

16.21.2.8 DynamicType∗ CDX::UnionDynamicType::get_selected_field (void)

Provides access to the field selected by the discriminator (or the default).

This is applicable only for union instances that have an initialized 'discriminator' value. This returns the field selected by the discriminator value (or by the 'default' value if the discriminator does not identify a field).

Return values

DynamicType *(p. 251)*	∗, or NULL if the discriminator is not associated with a field and there is no default.

16.21.2.9 DDS::ReturnCode_t CDX::UnionDynamicType::set_default_field (int32_t *field*)

Defines the index of the default field within the UNION.

The active field in the UNION is selected by considering the value of the discriminator and the values of the field labels. Any value of the discriminator not explicitly listed in the labels will select the default field. The default field may have other labels assigned to it, or it may have zero labels.

Note

An index value of -1 indicates that the UNION has no default field.

Return values

BAD_PARAMETER	if 't' is not of type UNION.
OK	upon success.

16.21.2.10 DDS::ReturnCode_t CDX::UnionDynamicType::set_discriminator (DynamicType ∗ *d*)

Defines the type and value of the UNION discriminator.

The discriminator is used to identify which one of the UNION fields is active. Only one field within the UNION is active at a given time. The active field is selected by considering the value of the discriminator and the values of the field labels. Each field can have a set of labels containing zero or more unique discriminator values. Field labels are defined with the **set_field_num_labels()** (p. 272) and **set_field_label()** (p. 272) functions.

Note

The UNION takes ownership of the provided discriminator.
One field in the union can be designated as the 'default' field. Any value of the discriminator not explicitly listed in the labels will select the default field. The default field may have other labels assigned to it, or it may have zero labels.

Return values

BAD_PARAMETER	if 't' is not of type UNION.
OK	upon success.

16.21.2.11 DDS::ReturnCode_t CDX::UnionDynamicType::set_field (uint32_t *n*, const char * *field_name*, DynamicType * *e*)

Assigns a value to a field of a STRUCT or UNION **DynamicType** (p. 251) object.

The 'nth' field of the struct or union is assigned the provided **field_name**, type **e**. This routine makes a copy of the 'field_name' argument.

Note

The STRUCT or UNION takes ownership of the provided field.
Fields of a UNION data type can not be part of a 'key set', as the field may not exist in a particular instantiation of the union data type.

Return values

BAD_PARAMETER	if 't' is not of type STRUCT or UNION.
OUT_OF_MEMORY	if memory allocation (to hold a copy of field_name) fails.
OK	upon success.

16.21.2.12 DDS::ReturnCode_t CDX::UnionDynamicType::set_field_label (uint32_t *field*, uint32_t *label*, int32_t *val*)

Assigns a label to the specified field within the UNION.

Return values

BAD_PARAMETER	if 't' is not of type UNION or if **field** does not specify a valid field index, or if **label** does not specify a valid label index for the field.
OUT_OF_RESOURCES	if memory allocation fails.
OK	upon success.

16.21.2.13 DDS::ReturnCode_t CDX::UnionDynamicType::set_field_num_labels (uint32_t *field*, uint32_t *n*)

Defines the number of labels associated with a field within the UNION.

Each field in a UNION has zero or more labels associated. The active field in the UNION is selected by considering the value of the discriminator and the values of the field labels. A field is selected when the value of the discriminator

matches the value of one of its labels. Any value of the discriminator not explicitly listed in the all of the labels will select the default field. [The default field may have other labels assigned to it, or it may have zero labels.]

Note

Subsequent calls to this routine for the same field will clear any previously assigned labels. [The default field may have other labels assigned to it, or it may have zero labels.]

Return values

BAD_PARAMETER	if 't' is not of type UNION.
OUT_OF_RESOURCES	if memory allocation fails.
OK	upon success.

16.21.2.14 DDS::ReturnCode_t CDX::UnionDynamicType::set_num_fields (uint32_t *n*)

Defines the number of fields held by a STRUCT or UNION **DynamicType** (p. 251) object.

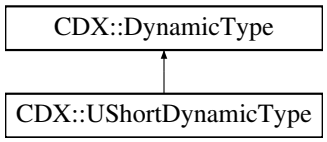
Return values

BAD_PARAMETER	if 't' is not of type STRUCT or UNION.
OK	upon success.

16.22 CDX::UShortDynamicType Class Reference

Contains an unsigned short.

Inheritance diagram for CDX::UShortDynamicType:



Public Member Functions

- uint16_t **get_ushort** ()

  *Provides access to data held in an UNSIGNED SHORT **DynamicType** (p. 251) object.*
- **DDS::ReturnCode_t set_ushort** (unsigned short c)

  *Assigns a value to the provided USHORT **DynamicType** (p. 251).*

16.22.1 Detailed Description

Contains an unsigned short.

16.22.2 Member Function Documentation

16.22.2.1 uint16_t CDX::UShortDynamicType::get_ushort ( )

Provides access to data held in an UNSIGNED SHORT **DynamicType** (p. 251) object.

Return values

uint16_t	the data value held by 't'.

16.22.2.2 DDS::ReturnCode_t CDX::UShortDynamicType::set_ushort (unsigned short *c*)

Assigns a value to the provided USHORT **DynamicType** (p. 251).

Return values

BAD_PARAMETER	if 't' is not of type USHORT.
OK	upon success.

Chapter 17

Data Structure Index

17.1 Class List

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

DDS::AnnotationDescriptor
A **AnnotationDescriptor** (p. 149) object comprises the state of an annotation as it is applied to some element . . . 149

CDX::ArrayDynamicType
Contains an array . . . 245

CDX::BooleanDynamicType
Contains a boolean . . . 248

DDS::BuiltinTopicKey_t . . . 131

CacheStatistics
Encapsulates statistics available from a DataReader or DataWriter . . . 141

CDX_XmlApi
Provides a generic XML Parser capability . . . 248

CDX::CharDynamicType
Contains a char . . . 249

DDS::rpc::ClientEndpoint . . . 205

DDS::rpc::ClientParams
Used to pass configuration parameters when constructing a Client . . . 206

DDS::Condition
A **Condition** (p. 23) can be added to a **WaitSet** (p. 73) to provide synchronous event notification . . . 23

DDS::ContentFilteredTopic
ContentFilteredTopic (p. 24) provides a topic that may exclude data based on a specified filter. The **ContentFilteredTopic** (p. 24) 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** (p. 26) associated with the **ContentFilteredTopic** (p. 24) . . . 24

CoreDX_LmtTransport
An instance of a Transport that can be added to a DomainParticipant . . . 195

CoreDX_LmtTransportConfig
Structure that holds LMT Transport configuration items . . . 196

CoreDX_SslTransport
An instance of a Transport that can be added to a DomainParticipant . . . **??**

CoreDX_SslTransportConfig
Structure that holds SSL Transport configuration items . . . **??**

CoreDX_TcpTransport
An instance of a Transport that can be added to a DomainParticipant . . . 197

CoreDX_TcpTransportConfig
Structure that holds TCP Transport configuration items . . . 198
CoreDX_UdpTransport
An instance of a Transport that can be added to a DomainParticipant . . . 199
CoreDX_UdpTransportConfig
Structure that holds UDP Transport configuration items . . . 200
DDS::DataReader
The **DataReader** (p. 26) entity allows the application to subscribe to and read data . . . 26
DDS::DataReaderListener
The **DataReaderListener** (p. 107) provides asynchronous notification of **DataReader** (p. 26) events . 107
DDS::DataReaderQos
Structure that holds **DataReader** (p. 26) Quality of Service policies . . . 75
DDS::DataRepresentationQosPolicy
Describes the data representation used by a topic . . . 83
DDS::DataWriter
The **DataWriter** (p. 37) entity provides an interface for the application to publish (write) data . . . 37
DDS::DataWriterListener
The **DataWriterListener** (p. 109) provides asynchronous notification of **DataWriter** (p. 37) events . . 109
DDS::DataWriterQos
Structure that holds **DataWriter** (p. 37) Quality of Service policies . . . 76
DDS::DCPSParticipant . . . 137
DDS::DCPSPublication . . . 137
DDS::DCPSSubscription . . . 139
DDS_Time_t . . . 135
DDS::DeadlineQosPolicy
This QoS policy establishes a minimum update period for data instances . . . 83
DDS::DestinationOrderQosPolicy
This QoS policy controls how each **Subscriber** (p. 65) orders received data samples . . . 84
DiscoveryQosPolicy
QoS Policy for configuring aspects of the Discovery and Builtin entities . . . 85
DDS::DomainEntity
Base class for all DDS Domain Entities . . . 43
DDS::DomainParticipant
The **DomainParticipant** (p. 44) is used to configure, create and destroy **Publisher** (p. 57), **Subscriber** (p. 65) and **Topic** (p. 69) objects . . . 44
DDS::DomainParticipantFactory
DomainParticipantFactory (p. 54) constructs DomainParticipants. The . . . 54
DDS::DomainParticipantFactoryQos
Structure that holds **DomainParticipantFactory** (p. 54) Quality of Service policies . . . 77
DDS::DomainParticipantListener
The **DomainParticipantListener** (p. 110) provides asynchronous notification of **DomainParticipant** (p. 44) events . . . 110
DDS::DomainParticipantQos
Structure that holds **DomainParticipant** (p. 44) Quality of Service policies . . . 78
CDX::DoubleDynamicType
Contains a double . . . 250
DDS::DurabilityQosPolicy
The **DurabilityQosPolicy** (p. 85) controls the durablity of data . . . 85
DDS::DurabilityServiceQosPolicy . . . 86
DDS::Duration_t
The **Duration_t** (p. 131) structure contains data to define a time duration . . . 131
DDS::DynamicData
A **DynamicData** (p. 151) object represents an individual data sample. It provides reflective getters and setters for the members of that sample . . . 151

DDS::DynamicDataFactory
This type is responsible for creating **DynamicData** (p. 151) instances . . . 168
DDS::DynamicDataReader
Provides a **DataReader** (p. 26) interface that is tailored to support reading an X-Types defined **DynamicData** (p. 151) data type. The specific **DynamicType** (p. 170) must have been registered previously with the **DomainParticipant** (p. 44) . . . 169
DDS::DynamicDataWriter
A **DataWriter** (p. 37) that writes '**DynamicData** (p. 151)' data elements. Data written by this **DataWriter** (p. 37) can be read by a standard **DataReader** (p. 26) or by a **DynamicDataReader** (p. 169), if the QoS and data types match . . . 169
DDS::DynamicType
An instance of **DynamicType** (p. 170) represent a type's schema: its physical name, kind, member definitions (if any), and so on . . . 170
CDX::DynamicType
DynamicType (p. 251) is an object that enhances CoreDX DDS with the facilities to process dynamic data types (in other words, data types that are not defined at compile time) . . . 251
DDS::DynamicTypeBuilder
A **DynamicTypeBuilder** (p. 173) object represents the state of a particular type defined according to the Type System. It is used to instantiate concrete **DynamicType** (p. 170) objects . . . 173
DDS::DynamicTypeBuilderFactory
An instance of this type is responsible for creating **DynamicType** (p. 170) and **DynamicTypeSupport** (p. 188) objects . . . 178
CDX::DynamicTypeDataReader
A DataReader that reads '**DynamicType** (p. 251)' data elements A **DynamicTypeDataReader** (p. 252) can read data published by a normal DataWriter or by a **DynamicTypeDataWriter** (p. 252), if the QoS and data types match . . . 252
CDX::DynamicTypeDataWriter
A DataWriter that writes '**DynamicType** (p. 251)' data elements. Data written by this DataWriter can be read by a standard DataReader or by a **DynamicTypeDataReader** (p. 252), if the QoS and data types match . . . 252
DDS::DynamicTypeMember
A **DynamicTypeMember** (p. 186) represents a "member" of a type. A "member" in this sense may be a member of an aggregated type, a constant within an enumeration, or some other type substructure 186
DDS::DynamicTypeSupport
The **DynamicTypeSupport** (p. 188) interface extends the TypeSupport interface defined by the **DDS** (p. 313) specification. This TypeSupport operates on **DynamicData** (p. 151) samples . . . 188
DDS::Entity
Base class for all DDS Entities . . . 55
EntityFactoryQosPolicy . . . 87
EntityNameQosPolicy . . . 88
CDX::EnumConstant
Represents an enum constant value . . . 253
CDX::EnumDynamicType
Represents an enum (set of constants) . . . 253
CDX::FloatDynamicType
Contains a float . . . 255
DDS::rpc::example::Foo . . . 233
DDS::rpc::example::Foo_Call . . . 234
DDS::rpc::example::Foo_op1_In . . . 235
DDS::rpc::example::Foo_op1_Out . . . 235
DDS::rpc::example::Foo_op1_Result . . . 236
DDS::rpc::example::Foo_op2_In . . . 237
DDS::rpc::example::Foo_op2_Out . . . 237
DDS::rpc::example::Foo_op2_Result . . . 238

DDS::rpc::example::Foo_Reply . . . 239
DDS::rpc::example::Foo_Request . . . 239
DDS::rpc::example::Foo_Return . . . 240
DDS::rpc::example::FooClient . . . 241
DDS::rpc::example::FooService . . . 242
DDS::GroupDataQosPolicy
Allows the application to attach arbitrary information to a **Publisher** (p. 57) or **Subscriber** (p. 65) . . . 88
DDS::GuardCondition
A **GuardCondition** (p. 56) is a condition where the **trigger_value** is under application control . . . 56
DDS::GUID_t . . . 132
DDS::HistoryQosPolicy
Controls the ammount of historical data maintained by a **DataReader** (p. 26) or **DataWriter** (p. 37) . . . 88
InconsistentTopicStatus
Status related to the on_inconsistent_topic listener methods of the DDS_TopicListener structure . . . 113
DDS::LatencyBudgetQosPolicy
Specifies allowable latency . . . 89
DDS::LifespanQosPolicy
Specifies the maximum duration of validity of the data written by the **DataWriter** (p. 37) . . . 90
DDS::rpc::ListenerBase
Basis for all rpc related listener classes . . . 207
LivelinessChangedStatus
Status related to the on_liveliness_changed listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures . . . 113
LivelinessLostStatus
Status related to the on_liveliness_lost listener methods of the DDS_DataWriter, DDS_Publisher, and DDS_DomainParticipant structures . . . 114
DDS::LivelinessQosPolicy
Determines the mechanism and parameters used by the application to determine whether an **Entity** (p. 55) is alive . . . 90
DDS::LoanedSamples< T >
Holds a collection of sample data and related meta-data . . . 119
Locator
Network address . . . 142
LoggingQosPolicy
Controls the amount and kind of information that is logged . . . 91
CDX::LongDynamicType
Contains a long . . . 255
CDX::LongLongDynamicType
Contains a long long . . . 256
DDS::MemberDescriptor
A **MemberDescriptor** (p. 190) comprises the state of a **DynamicTypeMember** (p. 186) . . . 190
CDX::OctetDynamicType
Contains an octet . . . 257
OfferedDeadlineMissedStatus
Status related to the on_offered_deadline_missed listener methods of the DDS_DataWriter, DDS_Publisher, and DDS_DomainParticipant structures . . . 114
OfferedIncompatibleQosStatus
Status related to the on_offered_incompatible_qos listener methods of the DDS_DataWriter, DDS_Publisher, and DDS_DomainParticipant structures . . . 115
DDS::OwnershipQosPolicy
Determines instance ownership in the case of multple writers. CoreDX DDS supports both SHARED_OWNERSHIP_QOS and EXCLUSIVE_OWNERSHIP_QOS . . . 92

DDS::OwnershipStrengthQosPolicy
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 92
ParticipantLocator
Describes a the location and identity of a potential peer DomainParticipant 142
DDS::PartitionQosPolicy
Defines a logical data partition 93
PeerParticipantQosPolicy
Configures a list of DomainParticipant peers to attempt communication with 94
DDS::PresentationQosPolicy
Controls the presentation of received data samples to the application. CoreDX DDS currently supports only the access_scope = INSTANCE_PRESENTATION_QOS policy 94
DDS::Property_t
A name-value pair property. The 'propagate' flag indicates if this property should be transfered through discovery 132
DDS::PropertyQosPolicy 95
PublicationMatchedStatus
Status related to the on_publication_matched listener methods of the DDS_DataWriter, DDS_Publisher, and DDS_DomainParticipant structures 115
DDS::Publisher
The **Publisher** (p. 57) configures, creates, manages and destroys DataWriters 57
DDS::PublisherListener
The **PublisherListener** (p. 110) provides asynchronous notification of **Publisher** (p. 57) events . . . 110
DDS::PublisherQos
Structure that holds **Publisher** (p. 57) Quality of Service policies 78
QosPolicyCount
Holds a DDS_QosPolicyId_t value and a counter to indicate the number of events associated with that PolicyId 143
DDS::QosProvider
QosProvider (p. 105) loads QoS settings from a library and provides interfaces to access entity specific QoS Policies 105
DDS::QueryCondition
A **QueryCondition** (p. 61) is a specialized **ReadCondition** (p. 63) which includes a filter 61
DDS::ReadCondition
A **ReadCondition** (p. 63) is a specialized **Condition** (p. 23) associated with a **DataReader** (p. 26) . . 63
ReaderDataLifecycleQosPolicy
Specifies the lifecycle behavior of data instances managed by the DataReader 95
DDS::ReliabilityQosPolicy
Indicates the level of reliability offered/provided by the **Entity** (p. 55). 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 96
DDS::rpc::Replier< TReq, TRep >
A replier receives requests and send replies 207
DDS::rpc::ReplierBase
Provides a generic basis for all **Replier** (p. 207) objects 211
DDS::rpc::ReplierListener< TReq, TRep >
ReplierLister can be installed on a **Replier** (p. 207) 212
DDS::rpc::ReplierParams
Used to pass configuration parameters when constructing a **Replier** (p. 207) 212
RequestedDeadlineMissedStatus
Status related to the on_requested_deadline_missed listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures 116

RequestedIncompatibleQosStatus
Status related to the on_requested_incompatible_qos listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures . . . 116
DDS::rpc::Requester< TReq, TRep >
A **Requester** (p. 214) sends requests and receives replies . . . 214
DDS::rpc::RequesterListener< TReq, TRep >
RequesterListener (p. 221) can be installed on a **Requester** (p. 214) . . . 221
DDS::rpc::RequesterParams
Used to pass configuration parameters when constructing a **Requester** (p. 214) . . . 222
DDS::ResourceLimitsQosPolicy
Specifies the resources that the Service can use to maintain data samples and instances . . . 97
DDS::rpc::RPCEntity
RPCEntity (p. 225) is a base abstract class . . . 225
DDS::RpcQosPolicy
Augment a **DataWriter** (p. 37) or **DataReader** (p. 26) with RPC specific information . . . 97
RTPSReaderQosPolicy
QoS Policy for configuring aspects of the RTPS Reader Protocol . . . 98
RTPSWriterQosPolicy
QoS Policy for configuring aspects of the RTPS Writer Protocol . . . 98
DDS::Sample< T >
Sample (p. 122) holds the data and related meta information . . . 122
DDS::SampleIdentity_t . . . 133
SampleInfo
The SampleInfo structure contains information associated with each sample . . . 143
DDS::SampleIterator< T, IsConst >
Provides a means to iterate over a collection of Samples . . . 123
SampleLostStatus
Status related to the on_sample_lost listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures . . . 116
DDS::SampleRef< T >
SampleRef (p. 125) holds a reference to data and related meta information . . . 125
SampleRejectedStatus
Status related to the on_sample_rejected listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures . . . 117
DDS::sequence< T >
Sequence data type . . . 145
CDX::SequenceDynamicType
Contains a sequence . . . 258
DDS::SequenceNumber_t . . . 134
DDS::rpc::ServiceEndpoint . . . 225
DDS::rpc::ServiceParams
Used to pass configuration parameters when constructing a Service . . . 227
DDS::rpc::ServiceProxy
ServiceProxy (p. 228) . . . 228
DDS::SharedSamples< T >
Holds a reference to a collection of sample data and related meta-data . . . 126
CDX::ShortDynamicType
Contains a short . . . 261
DDS::rpc::SimpleReplierListener< TReq, TRep >
SimpleReplierListener (p. 230) can be installed on a **Replier** (p. 207) . . . 230
DDS::rpc::SimpleRequesterListener< TRep >
SimpleReqeusterListener can be installed on a **Requester** (p. 214) . . . 231
DDS::StatusCondition
A **StatusCondition** (p. 64) is a condition associated with an **Entity** (p. 55) . . . 64

CDX::StringDynamicType
Contains a string . . . 262
CDX::StructDynamicType
Represents a struct of members. Each member has an associated data type . . . 264
DDS::Subscriber
The **Subscriber** (p. 65) configures, creates, manages and destroys DataReaders . . . 65
DDS::SubscriberListener
The **SubscriberListener** (p. 111) provides asynchronous notification of **Subscriber** (p. 65) events . . . 111
DDS::SubscriberQos
Structure that holds **Subscriber** (p. 65) Quality of Service policies . . . 79
SubscriptionMatchedStatus
Status related to the on_subscription_matched listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures . . . 117
ThreadModelQosPolicy
QoS Policy for configuring the threading behavior of the DomainParticipant . . . 99
DDS::Time_t
The **Time_t** (p. 134) structure contains data to define a time . . . 134
DDS::TimeBasedFilterQosPolicy
Defines a filter based on time between samples. The **DataReader** (p. 26) indicates that it wants at most one sample for each instance every minimum_separation interval . . . 99
DDS::Topic
Topic (p. 69) is the basic description of data to be published or subscribed . . . 69
DDS::TopicDataQosPolicy
Allows the application to attach arbitrary information to a **Topic** (p. 69) QoS . . . 100
DDS::TopicDescription
TopicDescription (p. 72) is an abstract 'class' that provides the foundation for **Topic** (p. 69), **ContentFilteredTopic** (p. 24), and MultiTopic . . . 72
DDS::TopicListener
The **TopicListener** (p. 112) provides asynchronous notification of **Topic** (p. 69) events . . . 112
DDS::TopicQos
Structure that holds **Topic** (p. 69) Quality of Service policies . . . 80
DDS::TransportPriorityQosPolicy
A hint to the middleware to help configure the transport priority mechanism . . . 101
DDS::TypecodeQosPolicy
Typecode representing the datatype a **DataReader** (p. 26) reads or a **DataWriter** (p. 37) writes . . . 101
DDS::TypeConsistencyEnforcementQosPolicy
Rules for determining type consistency . . . 102
DDS::TypeDescriptor
A **TypeDescriptor** (p. 191) comprises the state of a type . . . 191
CDX::ULongDynamicType
Contains an unsigned long . . . 266
CDX::ULongLongDynamicType
Contains an unsigned long long . . . 267
CDX::UnionDynamicType
Represents a union . . . 268
DDS::UserDataQosPolicy
Allows the application to attach arbitrary information to a **DomainParticipant** (p. 44), **DataWriter** (p. 37) or **DataReader** (p. 26) QoS . . . 102
CDX::UShortDynamicType
Contains an unsigned short . . . 273
DDS::WaitSet
A **WaitSet** (p. 73) maintains a set of **Condition** (p. 23) objects and allows the application to wait until one or more of them have a **trigger_value** of TRUE . . . 73

WriterDataLifecycleQosPolicy
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 103
DDS::WriteSample< T >
Holds sample data intended for transmission 128
DDS::WriteSampleRef< T >
Holds a reference to sample data intended for transmission 129

Chapter 18

File Documentation

18.1 cpp/dds.hh File Reference

CoreDX DDS C++ API Header File. The include/dds.hh file provides all of the CoreDX DDS declarations for C++ DDS datatypes and functions. This file should be included by C++ application code that uses the C++ DDS Application Programming Interface.

Classes

- struct **DDS::Time_t**

  *The **Time_t** (p. 134) structure contains data to define a time.*
- struct **EntityFactoryQosPolicy**
- struct **WriterDataLifecycleQosPolicy**

  *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.*
- struct **ReaderDataLifecycleQosPolicy**

  Specifies the lifecycle behavior of data instances managed by the DataReader.
- struct **EntityNameQosPolicy**
- struct **LoggingQosPolicy**

  Controls the amount and kind of information that is logged.
- struct **Locator**

  Network address.
- struct **ParticipantLocator**

  Describes a the location and identity of a potential peer DomainParticipant.
- struct **PeerParticipantQosPolicy**

  Configures a list of DomainParticipant peers to attempt communication with.
- struct **RTPSWriterQosPolicy**

  QoS Policy for configuring aspects of the RTPS Writer Protocol.
- struct **RTPSReaderQosPolicy**

  QoS Policy for configuring aspects of the RTPS Reader Protocol.
- struct **DiscoveryQosPolicy**

  QoS Policy for configuring aspects of the Discovery and Builtin entities.
- struct **ThreadModelQosPolicy**

QoS Policy for configuring the threading behavior of the DomainParticipant.

- class **DDS::DomainParticipantFactoryQos**

  *Structure that holds **DomainParticipantFactory** (p. 54) Quality of Service policies.*
- class **DDS::DomainParticipantQos**

  *Structure that holds **DomainParticipant** (p. 44) Quality of Service policies.*
- class **DDS::TopicQos**

  *Structure that holds **Topic** (p. 69) Quality of Service policies.*
- class **DDS::DataWriterQos**

  *Structure that holds **DataWriter** (p. 37) Quality of Service policies.*
- class **DDS::PublisherQos**

  *Structure that holds **Publisher** (p. 57) Quality of Service policies.*
- class **DDS::DataReaderQos**

  *Structure that holds **DataReader** (p. 26) Quality of Service policies.*
- class **DDS::SubscriberQos**

  *Structure that holds **Subscriber** (p. 65) Quality of Service policies.*
- class **DDS::Entity**

  Base class for all DDS Entities.
- class **DDS::DomainEntity**

  Base class for all DDS Domain Entities.
- class **DDS::DomainParticipantFactory**

  ***DomainParticipantFactory** (p. 54) constructs DomainParticipants. The.*
- class **DDS::DomainParticipant**

  *The **DomainParticipant** (p. 44) is used to configure, create and destroy **Publisher** (p. 57), **Subscriber** (p. 65) and **Topic** (p. 69) objects.*
- class **DDS::Publisher**

  *The **Publisher** (p. 57) configures, creates, manages and destroys DataWriters.*
- class **DDS::Subscriber**

  *The **Subscriber** (p. 65) configures, creates, manages and destroys DataReaders.*
- class **DDS::TopicDescription**

  ***TopicDescription** (p. 72) is an abstract 'class' that provides the foundation for **Topic** (p. 69), **ContentFilteredTopic** (p. 24), and MultiTopic.*
- class **DDS::Topic**

  ***Topic** (p. 69) is the basic description of data to be published or subscribed.*
- class **DDS::ContentFilteredTopic**

  ***ContentFilteredTopic** (p. 24) provides a topic that may exclude data based on a specified filter. The **ContentFilteredTopic** (p. 24) 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** (p. 26) associated with the **ContentFilteredTopic** (p. 24).*
- class **DDS::DataWriter**

  *The **DataWriter** (p. 37) entity provides an interface for the application to publish (write) data.*
- class **DDS::DataReader**

  *The **DataReader** (p. 26) entity allows the application to subscribe to and read data.*
- class **DDS::Condition**

  *A **Condition** (p. 23) can be added to a **WaitSet** (p. 73) to provide synchronous event notification.*
- class **DDS::GuardCondition**

  *A **GuardCondition** (p. 56) is a condition where the **trigger_value** is under application control.*
- class **DDS::StatusCondition**

  *A **StatusCondition** (p. 64) is a condition associated with an **Entity** (p. 55).*

- class **DDS::ReadCondition**

  *A **ReadCondition** (p. 63) is a specialized **Condition** (p. 23) associated with a **DataReader** (p. 26).*
- class **DDS::QueryCondition**

  *A **QueryCondition** (p. 61) is a specialized **ReadCondition** (p. 63) which includes a filter.*
- class **DDS::WaitSet**

  *A **WaitSet** (p. 73) maintains a set of **Condition** (p. 23) objects and allows the application to wait until one or more of them have a **trigger_value** of TRUE.*
- class **DDS::DataWriterListener**

  *The **DataWriterListener** (p. 109) provides asynchronous notification of **DataWriter** (p. 37) events.*
- class **DDS::PublisherListener**

  *The **PublisherListener** (p. 110) provides asynchronous notification of **Publisher** (p. 57) events.*
- class **DDS::TopicListener**

  *The **TopicListener** (p. 112) provides asynchronous notification of **Topic** (p. 69) events.*
- class **DDS::DataReaderListener**

  *The **DataReaderListener** (p. 107) provides asynchronous notification of **DataReader** (p. 26) events.*
- class **DDS::SubscriberListener**

  *The **SubscriberListener** (p. 111) provides asynchronous notification of **Subscriber** (p. 65) events.*
- class **DDS::DomainParticipantListener**

  *The **DomainParticipantListener** (p. 110) provides asynchronous notification of **DomainParticipant** (p. 44) events.*

Namespaces

- **DDS**

  Provides the DDS infrastructure.

Typedefs

- typedef DDS_DOMAINID_TYPE_NATIVE **DDS::DomainId_t**
- typedef DDS_HANDLE_TYPE_NATIVE **DDS::InstanceHandle_t**
- typedef long **DDS::ReturnCode_t**
- typedef unsigned long **DDS::StatusMask**
- typedef **DDS::sequence**< SampleInfo * > **DDS::SampleInfoSeq**

  sequence of SampleInfo pointers
- typedef **DDS::sequence**< DataReader * > **DDS::DataReaderSeq**

  *sequence of **DataReader** (p. 26) pointers*
- typedef **DDS::sequence**< InstanceHandle_t > **DDS::InstanceHandleSeq**

  sequence of InstanceHandle_t s
- typedef **DDS::sequence**< Condition * > **DDS::ConditionSeq**

  *sequence of **Condition** (p. 23) pointers*
- typedef **DDS::sequence**< void * > **DDS::Sequence**

  sequence of void pointers
- typedef **DDS::sequence**< char * > **DDS::StringSeq**

  sequence of character pointers (c strings)
- typedef **DDS::sequence**< unsigned char > **DDS::OctetSeq**

  sequence of octets

Enumerations

18.1.1 Detailed Description

CoreDX DDS C++ API Header File. The include/dds.hh file provides all of the CoreDX DDS declarations for C++ DDS datatypes and functions. This file should be included by C++ application code that uses the C++ DDS Application Programming Interface.

18.2 cpp/dds_builtin.hh File Reference

Classes

- struct **DDS::Duration_t**

  *The **Duration_t** (p. 131) structure contains data to define a time duration.*
- struct **DDS::UserDataQosPolicy**

  *Allows the application to attach arbitrary information to a **DomainParticipant** (p. 44), **DataWriter** (p. 37) or **DataReader** (p. 26) QoS.*
- struct **DDS::TopicDataQosPolicy**

  *Allows the application to attach arbitrary information to a **Topic** (p. 69) QoS.*
- struct **DDS::GroupDataQosPolicy**

  *Allows the application to attach arbitrary information to a **Publisher** (p. 57) or **Subscriber** (p. 65).*
- struct **DDS::TransportPriorityQosPolicy**

  A hint to the middleware to help configure the transport priority mechanism.
- struct **DDS::LifespanQosPolicy**

  *Specifies the maximum duration of validity of the data written by the **DataWriter** (p. 37).*
- struct **DDS::DurabilityQosPolicy**

  *The **DurabilityQosPolicy** (p. 85) controls the durablity of data.*
- struct **DDS::PresentationQosPolicy**

  Controls the presentation of received data samples to the application. CoreDX DDS currently supports only the access_scope = INSTANCE_PRESENTATION_QOS policy.
- struct **DDS::DeadlineQosPolicy**

  This QoS policy establishes a minimum update period for data instances.
- struct **DDS::LatencyBudgetQosPolicy**

  Specifies allowable latency.
- struct **DDS::OwnershipQosPolicy**

  Determines instance ownership in the case of multple writers. CoreDX DDS supports both SHARED_OWNERSHIP_QOS and EXCLUSIVE_OWNERSHIP_QOS.
- struct **DDS::OwnershipStrengthQosPolicy**

  *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.*
- struct **DDS::LivelinessQosPolicy**

  *Determines the mechanism and parameters used by the application to determine whether an **Entity** (p. 55) is alive.*
- struct **DDS::TimeBasedFilterQosPolicy**

  *Defines a filter based on time between samples. The **DataReader** (p. 26) indicates that it wants at most one sample for each instance every minimum_separation interval.*
- struct **DDS::PartitionQosPolicy**

  Defines a logical data partition.
- struct **DDS::ReliabilityQosPolicy**

  *Indicates the level of reliability offered/provided by the **Entity** (p. 55). 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.*
- struct **DDS::DestinationOrderQosPolicy**

  *This QoS policy controls how each **Subscriber** (p. 65) orders received data samples.*
- struct **DDS::HistoryQosPolicy**

  *Controls the ammount of historical data maintained by a **DataReader** (p. 26) or **DataWriter** (p. 37).*
- struct **DDS::ResourceLimitsQosPolicy**

Specifies the resources that the Service can use to maintain data samples and instances.

- struct **DDS::DurabilityServiceQosPolicy**
- struct **DDS::DataRepresentationQosPolicy**

  Describes the data representation used by a topic.
- struct **DDS::TypeConsistencyEnforcementQosPolicy**

  rules for determining type consistency
- struct **DDS::TypecodeQosPolicy**

  *Typecode representing the datatype a **DataReader** (p. 26) reads or a **DataWriter** (p. 37) writes.*
- struct **DDS::RpcQosPolicy**

  *Augment a **DataWriter** (p. 37) or **DataReader** (p. 26) with RPC specific information.*
- struct **DDS::Property_t**

  A name-value pair property. The 'propagate' flag indicates if this property should be transfered through discovery.
- struct **DDS::PropertyQosPolicy**
- struct **DDS::DCPSParticipant**
- struct **DDS::DCPSPublication**
- struct **DDS::DCPSSubscription**

Namespaces

- **DDS**

  Provides the DDS infrastructure.
- **DDS::rpc**

  Provides the RPC over DDS infrastructure.

Typedefs

- typedef short **DDS::DataRepresentationId_t**

  Indicates the form of data, for example CDR or XML.

Enumerations

18.3 cpp/dds_dtype.hh File Reference

CoreDX DDS DynamicTypes C++ API Header File.

Classes

- class **CDX::DynamicType**

  DynamicType *(p. 251) is an object that enhances CoreDX DDS with the facilities to process dynamic data types (in other words, data types that are not defined at compile time).*
- class **CDX::OctetDynamicType**

  Contains an octet.
- class **CDX::BooleanDynamicType**

  Contains a boolean.
- class **CDX::CharDynamicType**

  Contains a char.
- class **CDX::ShortDynamicType**

  Contains a short.
- class **CDX::UShortDynamicType**

  Contains an unsigned short.
- class **CDX::LongDynamicType**

  Contains a long.
- class **CDX::ULongDynamicType**

  Contains an unsigned long.
- class **CDX::LongLongDynamicType**

  Contains a long long.
- class **CDX::ULongLongDynamicType**

  Contains an unsigned long long.
- class **CDX::FloatDynamicType**

  Contains a float.
- class **CDX::DoubleDynamicType**

  Contains a double.
- class **CDX::StringDynamicType**

  Contains a string.
- class **CDX::ArrayDynamicType**

  Contains an array.
- class **CDX::SequenceDynamicType**

  Contains a sequence.
- class **CDX::EnumConstant**

  Represents an enum constant value.
- class **CDX::EnumDynamicType**

  Represents an enum (set of constants)
- class **CDX::StructDynamicType**

  Represents a struct of members. Each member has an associated data type.
- class **CDX::UnionDynamicType**

  Represents a union.
- class **CDX::DynamicTypeDataReader**

*A DataReader that reads '**DynamicType** (p. 251)' data elements A **DynamicTypeDataReader** (p. 252) can read data published by a normal DataWriter or by a **DynamicTypeDataWriter** (p. 252), if the QoS and data types match.*

- class **CDX::DynamicTypeDataWriter**

  *A DataWriter that writes '**DynamicType** (p. 251)' data elements. Data written by this DataWriter can be read by a standard DataReader or by a **DynamicTypeDataReader** (p. 252), if the QoS and data types match.*

Enumerations

18.3.1 Detailed Description

CoreDX DDS DynamicTypes C++ API Header File.

Deprecated NOTE: This API is deprecated. The preferred DynamicType API is provided in **xtypes_dtype.hh** (p. 301)

18.3.2 Enumeration Type Documentation

18.3.2.1 enum CDX::TypeCodeKind

Enumerator

TYPECODE_SHORT Typecode to identify a 'short' data type (16 bits with sign)

TYPECODE_LONG Typecode to identify a 'long' data type (32 bits with sign)

TYPECODE_USHORT Typecode to identify an 'unsigned short' data type (16 bits without sign)

TYPECODE_ULONG Typecode to identify an 'unsigned long' data type (32 bits without sign)

TYPECODE_FLOAT Typecode to identify a 'float' data type

TYPECODE_DOUBLE Typecode to identify a 'double' data type

TYPECODE_BOOLEAN Typecode to identify a 'boolean' data type

TYPECODE_CHAR Typecode to identify a 'char' data type (8 bits)

TYPECODE_OCTET Typecode to identify an 'octet' data type (8 bits)

TYPECODE_STRUCT Typecode to identify a 'struct' data type (with named sub-fields)

TYPECODE_UNION Typecode to identify a 'union' data type (with named sub-fields, selected by a discriminator)

TYPECODE_ENUM Typecode to identify an 'enum' data type

TYPECODE_STRING Typecode to identify a 'string' data type. May be bounded length or unlimited.

TYPECODE_SEQUENCE Typecode to identify a 'sequence' data type. May be bounded length or unlimited.

TYPECODE_ARRAY Typecode to identify an 'array' data type.

TYPECODE_ALIAS Not used. Included for compatibility.

TYPECODE_LONGLONG Typecode to identify a 'long long' data type (64 bits with sign)

TYPECODE_ULONGLONG Typecode to identify an 'unsigned long long' data type (64 bits without sign)

TYPECODE_LONGDOUBLE Typecode to identify a 'long double' data type. Not supported by CoreDX DDS.

TYPECODE_WCHAR Typecode to identify a 'wide char' data type. Not supported by CoreDX DDS.

TYPECODE_WSTRING Typecode to identify a 'wide string' data type. Not supported by CoreDX DDS.

TYPECODE_VALUE Not used. Included for compatibility.

TYPECODE_VALUE_PARAM Not used. Included for compatibility.

18.4 cpp/dds_seq.hh File Reference

CoreDX DDS C++ Sequence API Header File.

Classes

- class **DDS::sequence**< **T** >

  Sequence data type.

Namespaces

- **DDS**

  Provides the DDS infrastructure.

18.4.1 Detailed Description

CoreDX DDS C++ Sequence API Header File.

Provides an implementation of the Sequence data type. This data type is used in a few places throughout the DDS API.

It is most prevelant in the DataReader API where it is used to return a collection of samples and SampleInfo items back to the caller.

It is also used in a few QoS Policies.

18.5 cpp/dds_types.h File Reference

CoreDX DDS Basic Types C++ API Header File.

Classes

- struct **DDS_Time_t**
- struct **SampleInfo**

  The SampleInfo structure contains information associated with each sample.
- struct **InconsistentTopicStatus**

  Status related to the on_inconsistent_topic listener methods of the DDS_TopicListener structure.
- struct **SampleLostStatus**

  Status related to the on_sample_lost listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.
- struct **SampleRejectedStatus**

  Status related to the on_sample_rejected listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.
- struct **LivelinessLostStatus**

  Status related to the on_liveliness_lost listener methods of the DDS_DataWriter, DDS_Publisher, and DDS_DomainParticipant structures.
- struct **LivelinessChangedStatus**

  Status related to the on_liveliness_changed listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.
- struct **OfferedDeadlineMissedStatus**

  Status related to the on_offered_deadline_missed listener methods of the DDS_DataWriter, DDS_Publisher, and DDS_DomainParticipant structures.
- struct **RequestedDeadlineMissedStatus**

  Status related to the on_requested_deadline_missed listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.
- struct **QosPolicyCount**

  Holds a DDS_QosPolicyId_t value and a counter to indicate the number of events associated with that PolicyId.
- struct **OfferedIncompatibleQosStatus**

  Status related to the on_offered_incompatible_qos listener methods of the DDS_DataWriter, DDS_Publisher, and DDS_DomainParticipant structures.
- struct **RequestedIncompatibleQosStatus**

  Status related to the on_requested_incompatible_qos listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.
- struct **PublicationMatchedStatus**

  Status related to the on_publication_matched listener methods of the DDS_DataWriter, DDS_Publisher, and DDS_DomainParticipant structures.
- struct **SubscriptionMatchedStatus**

  Status related to the on_subscription_matched listener methods of the DDS_DataReader, DDS_Subscriber, and DDS_DomainParticipant structures.
- struct **CacheStatistics**

  Encapsulates statistics available from a DataReader or DataWriter.

Typedefs

- typedef DDS_DOMAINID_TYPE_NATIVE **DDS_DomainId_t**
- typedef DDS_HANDLE_TYPE_NATIVE **DDS_InstanceHandle_t**
- typedef int **DDS_ReturnCode_t**
- typedef unsigned int **DDS_StatusMask**

18.5.1 Detailed Description

CoreDX DDS Basic Types C++ API Header File.

18.5.2 Typedef Documentation

18.5.2.1 typedef DDS_DOMAINID_TYPE_NATIVE DDS_DomainId_t

Identifies the domain

18.5.2.2 typedef DDS_HANDLE_TYPE_NATIVE DDS_InstanceHandle_t

Identifies a data instance within the scope of a specific Entity.

18.5.2.3 typedef int DDS_ReturnCode_t

Indicates success or failure of an API operation.

18.5.2.4 typedef unsigned int DDS_StatusMask

A set of bits that select a set of statuses.

18.6 cpp/coredx_lmt_transport.h File Reference

CoreDX DDS Local Machine Transport (LMT)

Classes

- struct **CoreDX_LmtTransportConfig**

  Structure that holds LMT Transport configuration items.

18.6.1 Detailed Description

CoreDX DDS Local Machine Transport (LMT)

18.7 cpp/coredx_tcp_transport.h File Reference

CoreDX DDS TCP Transport.

Classes

- struct **CoreDX_TcpTransportConfig**

  Structure that holds TCP Transport configuration items.

18.7.1 Detailed Description

CoreDX DDS TCP Transport.

18.8 cpp/coredx_udp_transport.h File Reference

CoreDX DDS UDP Transport (default)

Classes

- struct **CoreDX_UdpTransportConfig**

  Structure that holds UDP Transport configuration items.

18.8.1 Detailed Description

CoreDX DDS UDP Transport (default)

18.9 cpp/function_call.hh File Reference

CoreDX DDS Remote Procedure Call (RPC) C++ Header File. This file defines the entites associated with the Function Call API.

Classes

- class **DDS::rpc::ServiceParams**

  Used to pass configuration parameters when constructing a Service.
- class **DDS::rpc::ClientParams**

  Used to pass configuration parameters when constructing a Client.
- class **DDS::rpc::ServiceEndpoint**
- class **DDS::rpc::ClientEndpoint**

Namespaces

- **DDS**

  Provides the DDS infrastructure.
- **DDS::rpc**

  Provides the RPC over DDS infrastructure.

Enumerations

18.9.1 Detailed Description

CoreDX DDS Remote Procedure Call (RPC) C++ Header File. This file defines the entites associated with the Function Call API.

18.10 cpp/qos_provider.hh File Reference

CoreDX DDS QosProvider C++ header file.

Classes

- class **DDS::QosProvider**

  QosProvider *(p. 105) loads QoS settings from a library and provides interfaces to access entity specific QoS Policies.*

Namespaces

- **DDS**

  Provides the DDS infrastructure.

18.10.1 Detailed Description

CoreDX DDS QosProvider C++ header file.

18.11 cpp/request_reply.hh File Reference

CoreDX DDS Remote Procedure Call (RPC) C++ Header File. This file defines the entites associated with the Request/Reply API.

Classes

- class **DDS::rpc::Requester**< **TReq, TRep** >

  *A **Requester** (p. 214) sends requests and receives replies.*
- class **DDS::rpc::Replier**< **TReq, TRep** >

  A replier receives requests and send replies.
- class **DDS::rpc::ListenerBase**

  Basis for all rpc related listener classes.
- class **DDS::rpc::SimpleReplierListener**< **TReq, TRep** >

  ***SimpleReplierListener** (p. 230) can be installed on a **Replier** (p. 207).*
- class **DDS::rpc::ReplierListener**< **TReq, TRep** >

  *ReplierLister can be installed on a **Replier** (p. 207).*
- class **DDS::rpc::SimpleRequesterListener**< **TRep** >

  *SimpleReqeusterListener can be installed on a **Requester** (p. 214).*
- class **DDS::rpc::RequesterListener**< **TReq, TRep** >

  ***RequesterListener** (p. 221) can be installed on a **Requester** (p. 214).*
- class **DDS::rpc::RPCEntity**

  ***RPCEntity** (p. 225) is a base abstract class.*
- class **DDS::rpc::ServiceProxy**

  ***ServiceProxy** (p. 228).*
- class **DDS::rpc::Requester**< **TReq, TRep** >

  *A **Requester** (p. 214) sends requests and receives replies.*
- class **DDS::rpc::ReplierBase**

  *Provides a generic basis for all **Replier** (p. 207) objects.*
- class **DDS::rpc::Replier**< **TReq, TRep** >

  A replier receives requests and send replies.
- class **DDS::rpc::RequesterParams**

  *Used to pass configuration parameters when constructing a **Requester** (p. 214).*
- class **DDS::rpc::ReplierParams**

  *Used to pass configuration parameters when constructing a **Replier** (p. 207).*

Namespaces

- **DDS**

  Provides the DDS infrastructure.
- **DDS::rpc**

  Provides the RPC over DDS infrastructure.

18.11.1 Detailed Description

CoreDX DDS Remote Procedure Call (RPC) C++ Header File. This file defines the entites associated with the Request/Reply API.

18.12 cpp/sample.hh File Reference

CoreDX **DDS** (p. 313) Sample<T> C++ Header File.

Classes

- class **DDS::SampleIterator< T, IsConst >**

  Provides a means to iterate over a collection of Samples.
- class **DDS::Sample< T >**

  Sample *(p. 122) holds the data and related meta information.*
- class **DDS::SampleRef< T >**

  SampleRef *(p. 125) holds a reference to data and related meta information.*
- class **DDS::WriteSample< T >**

  Holds sample data intended for transmission.
- class **DDS::WriteSampleRef< T >**

  Holds a reference to sample data intended for transmission.
- class **DDS::LoanedSamples< T >**

  Holds a collection of sample data and related meta-data.
- class **DDS::SharedSamples< T >**

  Holds a reference to a collection of sample data and related meta-data.
- class **DDS::rpc::Requester< TReq, TRep >**

  A ***Requester*** *(p. 214) sends requests and receives replies.*
- class **DDS::rpc::Replier< TReq, TRep >**

  A replier receives requests and send replies.
- class **DDS::Sample< T >**

  Sample *(p. 122) holds the data and related meta information.*
- class **DDS::SampleRef< T >**

  SampleRef *(p. 125) holds a reference to data and related meta information.*
- class **DDS::SampleIterator< T, IsConst >**

  Provides a means to iterate over a collection of Samples.
- class **DDS::LoanedSamples< T >**

  Holds a collection of sample data and related meta-data.
- class **DDS::SharedSamples< T >**

  Holds a reference to a collection of sample data and related meta-data.
- class **DDS::WriteSample< T >**

  Holds sample data intended for transmission.
- class **DDS::WriteSampleRef< T >**

  Holds a reference to sample data intended for transmission.

Namespaces

- **DDS**

  Provides the DDS infrastructure.
- **DDS::rpc**

  Provides the RPC over DDS infrastructure.

18.12.1 Detailed Description

CoreDX **DDS** (p. 313) Sample<T> C++ Header File.

18.13 cpp/xml_api.h File Reference

CoreDX DDS XML API provider header file.

Classes

- struct **CDX_XmlApi**

  Provides a generic XML Parser capability.

Variables

- **CDX_XmlApi cdx_libxml2_impl**

  the "dds_libxml2api" library provides this implementation that is based on libxml2
- **CDX_XmlApi cdx_xml_impl**

  the "dds_xml" library provides this internal XML parser implementation

18.13.1 Detailed Description

CoreDX DDS XML API provider header file.

18.14 cpp/xtypes.hh File Reference

Provides the X-Types DDS_DynamicDataReader and DDS_DynamicDataWriter API.

Classes

- class **DDS::DynamicDataReader**

  *Provides a **DataReader** (p. 26) interface that is tailored to support reading an X-Types defined **DynamicData** (p. 151) data type. The specific **DynamicType** (p. 170) must have been registered previously with the **DomainParticipant** (p. 44).*
- class **DDS::DynamicDataWriter**

  *A **DataWriter** (p. 37) that writes '**DynamicData** (p. 151)' data elements. Data written by this **DataWriter** (p. 37) can be read by a standard **DataReader** (p. 26) or by a **DynamicDataReader** (p. 169), if the QoS and data types match.*

Namespaces

- **DDS**

  Provides the DDS infrastructure.

Functions

- DDS::TypeObject * **DDS::TypecodeQosPolicy_to_typeobject** (TypecodeQosPolicy *tc_qos)

  Helper function to get DDS::TypeObject from a discovered (or otherwise obtained) TypeCode byte buffer.

18.14.1 Detailed Description

Provides the X-Types DDS_DynamicDataReader and DDS_DynamicDataWriter API.

18.15 cpp/xtypes_dtype.hh File Reference

Provides the X-Types DDS_DynamicType, DDS_DynamicData, and related APIs.

Classes

- class **DDS::DynamicTypeBuilderFactory**

  *An instance of this type is responsible for creating **DynamicType** (p. 170) and **DynamicTypeSupport** (p. 188) objects.*
- class **DDS::DynamicTypeSupport**

  *The **DynamicTypeSupport** (p. 188) interface extends the TypeSupport interface defined by the **DDS** (p. 313) specification. This TypeSupport operates on **DynamicData** (p. 151) samples.*
- class **DDS::AnnotationDescriptor**

  *A **AnnotationDescriptor** (p. 149) object comprises the state of an annotation as it is applied to some element.*
- class **DDS::TypeDescriptor**

  *A **TypeDescriptor** (p. 191) comprises the state of a type.*
- class **DDS::MemberDescriptor**

  *A **MemberDescriptor** (p. 190) comprises the state of a **DynamicTypeMember** (p. 186).*
- class **DDS::DynamicTypeMember**

  *A **DynamicTypeMember** (p. 186) represents a "member" of a type. A "member" in this sense may be a member of an aggregated type, a constant within an enumeration, or some other type substructure.*
- class **DDS::DynamicTypeBuilder**

  *A **DynamicTypeBuilder** (p. 173) object represents the state of a particular type defined according to the Type System. It is used to instantiate concrete **DynamicType** (p. 170) objects.*
- class **DDS::DynamicType**

  *An instance of **DynamicType** (p. 170) represent a type's schema: its physical name, kind, member definitions (if any), and so on.*
- class **DDS::DynamicDataFactory**

  *This type is responsible for creating **DynamicData** (p. 151) instances.*
- class **DDS::DynamicData**

  *A **DynamicData** (p. 151) object represents an individual data sample. It provides reflective getters and setters for the members of that sample.*

Namespaces

- **DDS**

  Provides the DDS infrastructure.

Functions

- DDS::TypeObject ∗ **DDS::TypecodeQosPolicy_to_typeobject** (TypecodeQosPolicy ∗tc_qos)

  Helper function to get DDS::TypeObject from a discovered (or otherwise obtained) TypeCode byte buffer.

18.15.1 Detailed Description

Provides the X-Types DDS_DynamicType, DDS_DynamicData, and related APIs.

Chapter 19

Not Yet Supported

Member DDS::AnnotationDescriptor::copy_from (p. 150) **(const class AnnotationDescriptor** (p. 149) **∗other)**

This operation is not yet implemented.

Member DDS::AnnotationDescriptor::equals (p. 150) **(const class AnnotationDescriptor** (p. 149) **∗other)**

This operation is not yet implemented.

Member DDS::AnnotationDescriptor::get_all_value (p. 150) **(Parameters ∗value)**

This operation is not yet implemented.

Member DDS::AnnotationDescriptor::get_value (p. 150) **(char ∗∗value, const char ∗key)**

This operation is not yet implemented.

Member DDS::AnnotationDescriptor::is_consistent (p. 150) **()**

This operation is not yet implemented.

Member DDS::AnnotationDescriptor::set_value (p. 150) **(const char ∗key, const char ∗value)**

This operation is not yet implemented.

Member DDS::DomainParticipant::create_multitopic (p. 47) **(const char ∗name, const char ∗type_name, const char ∗subscription_expression, const StringSeq &expression_params)**

This operation is not implemented yet.

Member DDS::DomainParticipant::delete_multitopic (p. 48) **(class MultiTopic ∗a_multitopic)**

This operation is not implemented yet.

Member DDS::DomainParticipant::ignore_publication (p. 52) **(InstanceHandle_t &handle)**

This operation is not implemented yet.

Member DDS::DomainParticipant::ignore_subscription (p. 52) **(InstanceHandle_t &handle)**

This operation is not implemented yet.

Member DDS::DomainParticipant::ignore_topic (p. 52) **(InstanceHandle_t &handle)**

This operation is not implemented yet.

Member DDS::DynamicData::get_boolean_values (p. 156) **(BooleanSeq ∗value, const MemberId id)=0**

This operation is not yet implemented.

Member DDS::DynamicData::get_byte_values (p. 157) **(ByteSeq ∗value, const MemberId id)=0**

This operation is not yet implemented.

Member DDS::DynamicData::get_char32_values (p. 157) **(WcharSeq ∗value, const MemberId id)=0**

This operation is not yet implemented.

Member DDS::DynamicData::get_char8_values (p. 157) **(CharSeq ∗value, const MemberId id)=0**

This operation is not yet implemented.

Member DDS::DynamicData::get_float32_values (p. 158) **(Float32Seq ∗value, const MemberId id)=0**

This operation is not yet implemented.

Member DDS::DynamicData::get_float64_values (p. 158) **(Float64Seq ∗value, const MemberId id)=0**

This operation is not yet implemented.

Member DDS::DynamicData::get_int16_values (p. 159) **(Int16Seq ∗value, const MemberId id)=0**

This operation is not yet implemented.

Member DDS::DynamicData::get_int32_values (p. 159) **(Int32Seq ∗value, const MemberId id)=0**

This operation is not yet implemented.

Member DDS::DynamicData::get_int64_values (p. 160) **(Int64Seq ∗value, const MemberId id)=0**

This operation is not yet implemented.

Member DDS::DynamicData::get_string_values (p. 160) **(StringSeq ∗value, const MemberId id)=0**

This operation is not yet implemented.

Member DDS::DynamicData::get_uint16_values (p. 161) **(UInt16Seq ∗value, const MemberId id)=0**

This operation is not yet implemented.

Member DDS::DynamicData::get_uint32_values (p. 161) **(UInt32Seq ∗value, const MemberId id)=0**

This operation is not yet implemented.

Member DDS::DynamicData::get_uint64_values (p. 161) **(UInt64Seq ∗value, const MemberId id)=0**

This operation is not yet implemented.

Member DDS::DynamicData::get_wstring_values (p. 162) **(WstringSeq ∗value, const MemberId id)=0**

This operation is not yet implemented.

Member DDS::DynamicData::set_boolean_values (p. 163) **(const MemberId id, const BooleanSeq ∗value)=0**

This operation is not yet implemented.

Member DDS::DynamicData::set_byte_values (p. 163) **(const MemberId id, const ByteSeq ∗value)=0**

This operation is not yet implemented.

Member DDS::DynamicData::set_char32_values (p. 163) **(const MemberId id, const WcharSeq ∗value)=0**

This operation is not yet implemented.

Member DDS::DynamicData::set_char8_values (p. 164) **(const MemberId id, const CharSeq ∗value)=0**

This operation is not yet implemented.

Member DDS::DynamicData::set_float32_values (p. 164) **(const MemberId id, const Float32Seq ∗value)=0**

This operation is not yet implemented.

Member DDS::DynamicData::set_float64_values (p. 165) **(const MemberId id, const Float64Seq ∗value)=0**

This operation is not yet implemented.

Member DDS::DynamicData::set_int16_values (p. 165) **(const MemberId id, const Int16Seq ∗value)=0**

This operation is not yet implemented.

Member DDS::DynamicData::set_int32_values (p. 165) **(const MemberId id, const Int32Seq ∗value)=0**

This operation is not yet implemented.

Member DDS::DynamicData::set_int64_values (p. 166) **(const MemberId id, const Int64Seq ∗value)=0**

This operation is not yet implemented.

Member DDS::DynamicData::set_string_values (p. 166) **(const MemberId id, const StringSeq ∗value)=0**

This operation is not yet implemented.

Member DDS::DynamicData::set_uint16_values (p. 167) **(const MemberId id, const UInt16Seq ∗value)=0**

This operation is not yet implemented.

Member DDS::DynamicData::set_uint32_values (p. 167) **(const MemberId id, const UInt32Seq ∗value)=0**

This operation is not yet implemented.

Member DDS::DynamicData::set_uint64_values (p. 167) **(const MemberId id, const UInt64Seq ∗value)=0**

This operation is not yet implemented.

Member DDS::DynamicData::set_wstring_values (p. 168) **(const MemberId id, const WstringSeq ∗value)=0**

This operation is not yet implemented.

Member DDS::DynamicType::get_annotation (p. 172) **(class AnnotationDescriptor** (p. 149) **∗descriptor, const unsigned int ∗idx)=0**

This operation is not yet implemented.

Member DDS::DynamicType::get_annotation_count (p. 172) **()=0**

This operation is not yet implemented.

Member DDS::DynamicTypeBuilder::apply_annotation (p. 175) **(const class AnnotationDescriptor** (p. 149) **∗descriptor)=0**

This operation is not yet implemented.

Member DDS::DynamicTypeBuilder::get_annotation (p. 176) **(class AnnotationDescriptor** (p. 149) **∗descriptor, const unsigned int idx)=0**

This operation is not yet implemented.

Member DDS::DynamicTypeBuilder::get_annotation_count (p. 176) **()=0**

This operation is not yet implemented.

Member DDS::DynamicTypeBuilderFactory::create_type_w_document (p. 183) **(const char ∗document, const char ∗type_name, const IncludePathSeq ∗include_paths)=0**

This operation is not yet implemented.

Member DDS::DynamicTypeBuilderFactory::create_type_w_uri (p. 183) **(const char ∗document_url, const char ∗type_name, const IncludePathSeq ∗include_paths)=0**

This operation is not yet implemented.

Member DDS::DynamicTypeMember::get_annotation (p. 187) **(class AnnotationDescriptor** (p. 149) **∗descriptor, const unsigned int idx)=0**

This operation is not yet implemented.

Member DDS::DynamicTypeMember::get_annotation_count (p. 187) **()=0**

This operation is not yet implemented.

Member DDS::Publisher::begin_coherent_changes (p. 58) **()**

This operation is not yet implemented.

Member DDS::Publisher::end_coherent_changes (p. 60) **()**

This operation is not yet implemented.

Member DDS::Publisher::resume_publications (p. 61) **()**

This operation is not yet implemented.

Member DDS::Publisher::suspend_publications (p. 61) **()**

This operation is not yet implemented.

Class DDS::QueryCondition (p. 61)

CoreDX DDS does not yet implement QueryConditions as a trigger for a **WaitSet** (p. 73).

Member DDS::rpc::Replier< TReq, TRep >::receive_nondata_samples (p. 209) **(bool enable)**

Member DDS::rpc::Requester< TReq, TRep >::wait_for_replies (p. 221) **(uint32_t min_count, const DDS::Duration_t** (p. 131) **&max_wait, const DDS::SampleIdentity_t** (p. 133) **&related_request_id)**

Currently, this routine will return if **at** least 'min_count' replies are available – it ignores 'related_request'.

Chapter 20

Deprecated List

File dds_dtype.hh (p. 289)

NOTE: This API is deprecated. The preferred DynamicType API is provided in **xtypes_dtype.hh** (p. 301)

Chapter 21

Hierarchical Index

21.1 Class Hierarchy

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

DDS::AnnotationDescriptor . . . 149
DDS::BuiltinTopicKey_t . . . 131
CacheStatistics . . . 141
CDX_XmlApi . . . 248
DDS::rpc::ClientParams . . . 206
DDS::Condition . . . 23
  DDS::GuardCondition . . . 56
  DDS::ReadCondition . . . 63
    DDS::QueryCondition . . . 61
  DDS::StatusCondition . . . 64
CoreDX_LmtTransport . . . 195
CoreDX_LmtTransportConfig . . . 196
CoreDX_SslTransport . . . **??**
CoreDX_SslTransportConfig . . . **??**
CoreDX_TcpTransport . . . 197
CoreDX_TcpTransportConfig . . . 198
CoreDX_UdpTransport . . . 199
CoreDX_UdpTransportConfig . . . 200
DDS::DataReaderListener . . . 107
  DDS::SubscriberListener . . . 111
    DDS::DomainParticipantListener . . . 110
DDS::DataReaderQos . . . 75
DDS::DataRepresentationQosPolicy . . . 83
DDS::DataWriterListener . . . 109
  DDS::PublisherListener . . . 110
    DDS::DomainParticipantListener . . . 110
DDS::DataWriterQos . . . 76
DDS::DCPSParticipant . . . 137
DDS::DCPSPublication . . . 137
DDS::DCPSSubscription . . . 139
DDS_Time_t . . . 135
DDS::DeadlineQosPolicy . . . 83
DDS::DestinationOrderQosPolicy . . . 84

DiscoveryQosPolicy . . . 85
DDS::DomainParticipantFactory . . . 54
DDS::DomainParticipantFactoryQos . . . 77
DDS::DomainParticipantQos . . . 78
DDS::DurabilityQosPolicy . . . 85
DDS::DurabilityServiceQosPolicy . . . 86
DDS::Duration_t . . . 131
DDS::DynamicData . . . 151
DDS::DynamicDataFactory . . . 168
DDS::DynamicType . . . 170
CDX::DynamicType . . . 251
- CDX::ArrayDynamicType . . . 245
- CDX::BooleanDynamicType . . . 248
- CDX::CharDynamicType . . . 249
- CDX::DoubleDynamicType . . . 250
- CDX::EnumDynamicType . . . 253
- CDX::FloatDynamicType . . . 255
- CDX::LongDynamicType . . . 255
- CDX::LongLongDynamicType . . . 256
- CDX::OctetDynamicType . . . 257
- CDX::SequenceDynamicType . . . 258
- CDX::ShortDynamicType . . . 261
- CDX::StringDynamicType . . . 262
- CDX::StructDynamicType . . . 264
- CDX::ULongDynamicType . . . 266
- CDX::ULongLongDynamicType . . . 267
- CDX::UnionDynamicType . . . 268
- CDX::UShortDynamicType . . . 273

DDS::DynamicTypeBuilder . . . 173
DDS::DynamicTypeBuilderFactory . . . 178
DDS::DynamicTypeMember . . . 186
DDS::DynamicTypeSupport . . . 188
DDS::Entity . . . 55
- DDS::DataReader . . . 26
  - CDX::DynamicTypeDataReader . . . 252
  - DDS::DynamicDataReader . . . 169
- DDS::DataWriter . . . 37
  - CDX::DynamicTypeDataWriter . . . 252
  - DDS::DynamicDataWriter . . . 169
- DDS::DomainEntity . . . 43
- DDS::DomainParticipant . . . 44
- DDS::Publisher . . . 57
- DDS::Subscriber . . . 65
- DDS::Topic . . . 69

EntityFactoryQosPolicy . . . 87
EntityNameQosPolicy . . . 88
CDX::EnumConstant . . . 253
DDS::rpc::example::Foo . . . 233
- DDS::rpc::example::FooClient . . . 241
- DDS::rpc::example::FooService . . . 242

DDS::rpc::example::Foo_Call . . . 234
DDS::rpc::example::Foo_op1_In . . . 235
DDS::rpc::example::Foo_op1_Out . . . 235

DDS::rpc::example::Foo_op1_Result . . . 236
DDS::rpc::example::Foo_op2_In . . . 237
DDS::rpc::example::Foo_op2_Out . . . 237
DDS::rpc::example::Foo_op2_Result . . . 238
DDS::rpc::example::Foo_Reply . . . 239
DDS::rpc::example::Foo_Request . . . 239
DDS::rpc::example::Foo_Return . . . 240
DDS::GroupDataQosPolicy . . . 88
DDS::GUID_t . . . 132
DDS::HistoryQosPolicy . . . 88
InconsistentTopicStatus . . . 113
DDS::LatencyBudgetQosPolicy . . . 89
DDS::LifespanQosPolicy . . . 90
DDS::rpc::ListenerBase . . . 207
  DDS::rpc::ReplierListener< TReq, TRep > . . . 212
  DDS::rpc::RequesterListener< TReq, TRep > . . . 221
  DDS::rpc::SimpleReplierListener< TReq, TRep > . . . 230
  DDS::rpc::SimpleRequesterListener< TRep > . . . 231
  DDS::rpc::ReplierListener< DDS::rpc::example::Foo_Request, DDS::rpc::example::Foo_Reply > . . . 212
  DDS::rpc::ReplierListener< Foo_Request, Foo_Reply > . . . 212
  DDS::rpc::SimpleReplierListener< DDS::rpc::example::Foo_Request, DDS::rpc::example::Foo_Reply > . . . 230
LivelinessChangedStatus . . . 113
LivelinessLostStatus . . . 114
DDS::LivelinessQosPolicy . . . 90
DDS::LoanedSamples< T > . . . 119
Locator . . . 142
LoggingQosPolicy . . . 91
DDS::MemberDescriptor . . . 190
OfferedDeadlineMissedStatus . . . 114
OfferedIncompatibleQosStatus . . . 115
DDS::OwnershipQosPolicy . . . 92
DDS::OwnershipStrengthQosPolicy . . . 92
ParticipantLocator . . . 142
DDS::PartitionQosPolicy . . . 93
PeerParticipantQosPolicy . . . 94
DDS::PresentationQosPolicy . . . 94
DDS::Property_t . . . 132
DDS::PropertyQosPolicy . . . 95
PublicationMatchedStatus . . . 115
DDS::PublisherQos . . . 78
QosPolicyCount . . . 143
DDS::QosProvider . . . 105
ReaderDataLifecycleQosPolicy . . . 95
DDS::ReliabilityQosPolicy . . . 96
DDS::rpc::ReplierParams . . . 212
RequestedDeadlineMissedStatus . . . 116
RequestedIncompatibleQosStatus . . . 116
DDS::rpc::RequesterParams . . . 222
DDS::ResourceLimitsQosPolicy . . . 97
DDS::rpc::RPCEntity . . . 225
  DDS::rpc::ReplierBase . . . 211
    DDS::rpc::Replier< TReq, TRep > . . . 207
    DDS::rpc::Replier< DDS::rpc::example::Foo_Request, DDS::rpc::example::Foo_Reply > . . . 207
  DDS::rpc::ServiceEndpoint . . . 225

DDS::rpc::example::FooService . . . 242
DDS::rpc::ServiceProxy . . . 228
DDS::rpc::ClientEndpoint . . . 205
DDS::rpc::example::FooClient . . . 241
DDS::rpc::Requester< TReq, TRep > . . . 214
DDS::RpcQosPolicy . . . 97
RTPSReaderQosPolicy . . . 98
RTPSWriterQosPolicy . . . 98
DDS::Sample< T > . . . 122
DDS::SampleIdentity_t . . . 133
SampleInfo . . . 143
DDS::SampleIterator< T, IsConst > . . . 123
SampleLostStatus . . . 116
DDS::SampleRef< T > . . . 125
SampleRejectedStatus . . . 117
DDS::sequence< T > . . . 145
DDS::sequence< char * > . . . 145
DDS::sequence< SampleInfo * > . . . 145
DDS::sequence< unsigned char > . . . 145
DDS::SequenceNumber_t . . . 134
DDS::rpc::ServiceParams . . . 227
DDS::SharedSamples< T > . . . 126
DDS::SubscriberQos . . . 79
SubscriptionMatchedStatus . . . 117
ThreadModelQosPolicy . . . 99
DDS::Time_t . . . 134
DDS::TimeBasedFilterQosPolicy . . . 99
DDS::TopicDataQosPolicy . . . 100
DDS::TopicDescription . . . 72
DDS::ContentFilteredTopic . . . 24
DDS::Topic . . . 69
DDS::TopicListener . . . 112
DDS::DomainParticipantListener . . . 110
DDS::TopicQos . . . 80
DDS::TransportPriorityQosPolicy . . . 101
DDS::TypecodeQosPolicy . . . 101
DDS::TypeConsistencyEnforcementQosPolicy . . . 102
DDS::TypeDescriptor . . . 191
DDS::UserDataQosPolicy . . . 102
DDS::WaitSet . . . 73
WriterDataLifecycleQosPolicy . . . 103
DDS::WriteSample< T > . . . 128
DDS::WriteSampleRef< T > . . . 129

Chapter 22

Namespace Details

22.1 Namespace List

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

DDS
Provides the DDS infrastructure 313
DDS::rpc
Provides the RPC over DDS infrastructure 320
DDS::rpc::example
Provides an example of generated RPC over DDS code. The structures are generated for this IDL: . 321

22.2 DDS Namespace Reference

Provides the DDS infrastructure.

Namespaces

- **rpc**

  Provides the RPC over DDS infrastructure.

Classes

- class **AnnotationDescriptor**

  *A **AnnotationDescriptor** (p. 149) object comprises the state of an annotation as it is applied to some element.*
- struct **BuiltinTopicKey_t**
- class **Condition**

  *A **Condition** (p. 23) can be added to a **WaitSet** (p. 73) to provide synchronous event notification.*
- class **ContentFilteredTopic**

  ***ContentFilteredTopic** (p. 24) provides a topic that may exclude data based on a specified filter. The **ContentFilteredTopic** (p. 24) 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** (p. 26) associated with the **ContentFilteredTopic** (p. 24).*
- class **DataReader**

*The **DataReader** (p. 26) entity allows the application to subscribe to and read data.*

- class **DataReaderListener**

  *The **DataReaderListener** (p. 107) provides asynchronous notification of **DataReader** (p. 26) events.*

- class **DataReaderQos**

  *Structure that holds **DataReader** (p. 26) Quality of Service policies.*

- struct **DataRepresentationQosPolicy**

  Describes the data representation used by a topic.

- class **DataWriter**

  *The **DataWriter** (p. 37) entity provides an interface for the application to publish (write) data.*

- class **DataWriterListener**

  *The **DataWriterListener** (p. 109) provides asynchronous notification of **DataWriter** (p. 37) events.*

- class **DataWriterQos**

  *Structure that holds **DataWriter** (p. 37) Quality of Service policies.*

- struct **DCPSParticipant**
- struct **DCPSPublication**
- struct **DCPSSubscription**
- struct **DeadlineQosPolicy**

  This QoS policy establishes a minimum update period for data instances.

- struct **DestinationOrderQosPolicy**

  *This QoS policy controls how each **Subscriber** (p. 65) orders received data samples.*

- class **DomainEntity**

  Base class for all DDS Domain Entities.

- class **DomainParticipant**

  *The **DomainParticipant** (p. 44) is used to configure, create and destroy **Publisher** (p. 57), **Subscriber** (p. 65) and **Topic** (p. 69) objects.*

- class **DomainParticipantFactory**

  ***DomainParticipantFactory** (p. 54) constructs DomainParticipants. The.*

- class **DomainParticipantFactoryQos**

  *Structure that holds **DomainParticipantFactory** (p. 54) Quality of Service policies.*

- class **DomainParticipantListener**

  *The **DomainParticipantListener** (p. 110) provides asynchronous notification of **DomainParticipant** (p. 44) events.*

- class **DomainParticipantQos**

  *Structure that holds **DomainParticipant** (p. 44) Quality of Service policies.*

- struct **DurabilityQosPolicy**

  *The **DurabilityQosPolicy** (p. 85) controls the durablity of data.*

- struct **DurabilityServiceQosPolicy**
- struct **Duration_t**

  *The **Duration_t** (p. 131) structure contains data to define a time duration.*

- class **DynamicData**

  *A **DynamicData** (p. 151) object represents an individual data sample. It provides reflective getters and setters for the members of that sample.*

- class **DynamicDataFactory**

  *This type is responsible for creating **DynamicData** (p. 151) instances.*

- class **DynamicDataReader**

  *Provides a **DataReader** (p. 26) interface that is tailored to support reading an X-Types defined **DynamicData** (p. 151) data type. The specific **DynamicType** (p. 170) must have been registered previously with the **DomainParticipant** (p. 44).*

- class **DynamicDataWriter**

*A **DataWriter** (p. 37) that writes '**DynamicData** (p. 151)' data elements. Data written by this **DataWriter** (p. 37) can be read by a standard **DataReader** (p. 26) or by a **DynamicDataReader** (p. 169), if the QoS and data types match.*

- class **DynamicType**

  *An instance of **DynamicType** (p. 170) represent a type's schema: its physical name, kind, member definitions (if any), and so on.*

- class **DynamicTypeBuilder**

  *A **DynamicTypeBuilder** (p. 173) object represents the state of a particular type defined according to the Type System. It is used to instantiate concrete **DynamicType** (p. 170) objects.*

- class **DynamicTypeBuilderFactory**

  *An instance of this type is responsible for creating **DynamicType** (p. 170) and **DynamicTypeSupport** (p. 188) objects.*

- class **DynamicTypeMember**

  *A **DynamicTypeMember** (p. 186) represents a "member" of a type. A "member" in this sense may be a member of an aggregated type, a constant within an enumeration, or some other type substructure.*

- class **DynamicTypeSupport**

  *The **DynamicTypeSupport** (p. 188) interface extends the TypeSupport interface defined by the **DDS** (p. 313) specification. This TypeSupport operates on **DynamicData** (p. 151) samples.*

- class **Entity**

  Base class for all DDS Entities.

- struct **GroupDataQosPolicy**

  *Allows the application to attach arbitrary information to a **Publisher** (p. 57) or **Subscriber** (p. 65).*

- class **GuardCondition**

  *A **GuardCondition** (p. 56) is a condition where the **trigger_value** is under application control.*

- struct **GUID_t**
- struct **HistoryQosPolicy**

  *Controls the ammount of historical data maintained by a **DataReader** (p. 26) or **DataWriter** (p. 37).*

- struct **LatencyBudgetQosPolicy**

  Specifies allowable latency.

- struct **LifespanQosPolicy**

  *Specifies the maximum duration of validity of the data written by the **DataWriter** (p. 37).*

- struct **LivelinessQosPolicy**

  *Determines the mechanism and parameters used by the application to determine whether an **Entity** (p. 55) is alive.*

- class **LoanedSamples**

  Holds a collection of sample data and related meta-data.

- class **MemberDescriptor**

  *A **MemberDescriptor** (p. 190) comprises the state of a **DynamicTypeMember** (p. 186).*

- struct **OwnershipQosPolicy**

  Determines instance ownership in the case of multple writers. CoreDX DDS supports both SHARED_OWNERSHIP_QOS and EXCLUSIVE_OWNERSHIP_QOS.

- struct **OwnershipStrengthQosPolicy**

  *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.*

- struct **PartitionQosPolicy**

  Defines a logical data partition.

- struct **PresentationQosPolicy**

  Controls the presentation of received data samples to the application. CoreDX DDS currently supports only the access_scope = INSTANCE_PRESENTATION_QOS policy.

- struct **Property_t**

  A name-value pair property. The 'propagate' flag indicates if this property should be transfered through discovery.

- struct **PropertyQosPolicy**
- class **Publisher**

  *The **Publisher** (p. 57) configures, creates, manages and destroys DataWriters.*
- class **PublisherListener**

  *The **PublisherListener** (p. 110) provides asynchronous notification of **Publisher** (p. 57) events.*
- class **PublisherQos**

  *Structure that holds **Publisher** (p. 57) Quality of Service policies.*
- class **QosProvider**

  ***QosProvider** (p. 105) loads QoS settings from a library and provides interfaces to access entity specific QoS Policies.*
- class **QueryCondition**

  *A **QueryCondition** (p. 61) is a specialized **ReadCondition** (p. 63) which includes a filter.*
- class **ReadCondition**

  *A **ReadCondition** (p. 63) is a specialized **Condition** (p. 23) associated with a **DataReader** (p. 26).*
- struct **ReliabilityQosPolicy**

  *Indicates the level of reliability offered/provided by the **Entity** (p. 55). 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.*
- struct **ResourceLimitsQosPolicy**

  Specifies the resources that the Service can use to maintain data samples and instances.
- struct **RpcQosPolicy**

  *Augment a **DataWriter** (p. 37) or **DataReader** (p. 26) with RPC specific information.*
- class **Sample**

  ***Sample** (p. 122) holds the data and related meta information.*
- struct **SampleIdentity_t**
- class **SampleIterator**

  Provides a means to iterate over a collection of Samples.
- class **SampleRef**

  ***SampleRef** (p. 125) holds a reference to data and related meta information.*
- class **sequence**

  Sequence data type.
- struct **SequenceNumber_t**
- class **SharedSamples**

  Holds a reference to a collection of sample data and related meta-data.
- class **StatusCondition**

  *A **StatusCondition** (p. 64) is a condition associated with an **Entity** (p. 55).*
- class **Subscriber**

  *The **Subscriber** (p. 65) configures, creates, manages and destroys DataReaders.*
- class **SubscriberListener**

  *The **SubscriberListener** (p. 111) provides asynchronous notification of **Subscriber** (p. 65) events.*
- class **SubscriberQos**

  *Structure that holds **Subscriber** (p. 65) Quality of Service policies.*
- struct **Time_t**

  *The **Time_t** (p. 134) structure contains data to define a time.*
- struct **TimeBasedFilterQosPolicy**

  *Defines a filter based on time between samples. The **DataReader** (p. 26) indicates that it wants at most one sample for each instance every minimum_separation interval.*
- class **Topic**

  ***Topic** (p. 69) is the basic description of data to be published or subscribed.*
- struct **TopicDataQosPolicy**

*Allows the application to attach arbitrary information to a **Topic** (p. 69) QoS.*

- class **TopicDescription**

  *__TopicDescription__ (p. 72) is an abstract 'class' that provides the foundation for **Topic** (p. 69), **ContentFilteredTopic** (p. 24), and MultiTopic.*
- class **TopicListener**

  *The **TopicListener** (p. 112) provides asynchronous notification of **Topic** (p. 69) events.*
- class **TopicQos**

  *Structure that holds **Topic** (p. 69) Quality of Service policies.*
- struct **TransportPriorityQosPolicy**

  A hint to the middleware to help configure the transport priority mechanism.
- struct **TypecodeQosPolicy**

  *Typecode representing the datatype a **DataReader** (p. 26) reads or a **DataWriter** (p. 37) writes.*
- struct **TypeConsistencyEnforcementQosPolicy**

  rules for determining type consistency
- class **TypeDescriptor**

  *A **TypeDescriptor** (p. 191) comprises the state of a type.*
- struct **UserDataQosPolicy**

  *Allows the application to attach arbitrary information to a **DomainParticipant** (p. 44), **DataWriter** (p. 37) or **DataReader** (p. 26) QoS.*
- class **WaitSet**

  *A **WaitSet** (p. 73) maintains a set of **Condition** (p. 23) objects and allows the application to wait until one or more of them have a **trigger_value** of TRUE.*
- class **WriteSample**

  Holds sample data intended for transmission.
- class **WriteSampleRef**

  Holds a reference to sample data intended for transmission.

Typedefs

- typedef unsigned char **GuidPrefix_t**[12]
- typedef short **DataRepresentationId_t**

  Indicates the form of data, for example CDR or XML.
- typedef DDS_DOMAINID_TYPE_NATIVE **DomainId_t**
- typedef DDS_HANDLE_TYPE_NATIVE **InstanceHandle_t**
- typedef long **ReturnCode_t**
- typedef unsigned long **StatusMask**
- typedef **DDS::sequence**< SampleInfo * > **SampleInfoSeq**

  sequence of SampleInfo pointers
- typedef **DDS::sequence**< **DataReader** * > **DataReaderSeq**

  *sequence of **DataReader** (p. 26) pointers*
- typedef **DDS::sequence**< **InstanceHandle_t** > **InstanceHandleSeq**

  sequence of InstanceHandle_t s
- typedef **DDS::sequence**< **Condition** * > **ConditionSeq**

  *sequence of **Condition** (p. 23) pointers*
- typedef **DDS::sequence**< void * > **Sequence**

  sequence of void pointers
- typedef **DDS::sequence**< char * > **StringSeq**

  sequence of character pointers (c strings)
- typedef **DDS::sequence**< unsigned char > **OctetSeq**

  sequence of octets

Enumerations

Functions

- DDS::TypeObject * **TypecodeQosPolicy_to_typeobject** (**TypecodeQosPolicy** *tc_qos)

  Helper function to get DDS::TypeObject from a discovered (or otherwise obtained) TypeCode byte buffer.

22.2.1 Detailed Description

Provides the DDS infrastructure.

namespace **DDS** (p. 313)

22.2.2 Typedef Documentation

22.2.2.1 typedef DDS_DOMAINID_TYPE_NATIVE DDS::DomainId_t

Identifies the domain

22.2.2.2 typedef unsigned char DDS::GuidPrefix_t[12]

Another form of builtin topic key (12 bytes).

22.2.2.3 typedef DDS_HANDLE_TYPE_NATIVE DDS::InstanceHandle_t

Identifies a data instance within the scope of a specific **Entity** (p. 55).

22.2.2.4 typedef long DDS::ReturnCode_t

Indicates success or failure of an API operation.

22.2.2.5 typedef unsigned long DDS::StatusMask

A set of bits that select a set of statuses.

22.2.3 Enumeration Type Documentation

22.2.3.1 enum DDS::DestinationOrderQosPolicyKind

This enumeration contains the kinds of Destination Ordering.

Enumerator

BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS Use the reception time to determine the order of samples.

BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS Use the timestamp set by the publisher to determine the order of samples.

22.2.3.2 enum DDS::DiscoveryQosPolicyDiscoveryKind

This enumeration contains the kinds of Discovery.

Enumerator

PEER_DISCOVERY_QOS The **DomainParticipant** (p. 44) will use the standard peer-to-peer discovery.

CENTRAL_DISCOVERY_QOS The **DomainParticipant** (p. 44) will connect to the CoreDX DDS CDD for discovery.

22.2.3.3 enum DDS::DurabilityQosPolicyKind

This enumeration contains the kinds of Durability.

Enumerator

VOLATILE_DURABILITY_QOS The data is volatile, and is not persisted beyond the initial publication.

TRANSIENT_LOCAL_DURABILITY_QOS The data is persisted locally within the source **DataWriter** (p. 37).

TRANSIENT_DURABILITY_QOS The data is persisted beyond the lifecycle of the originating **DataWriter** (p. 37), and is available even after that **DataWriter** (p. 37) has been destroyed.

PERSISTENT_DURABILITY_QOS The data is persisted to 'off-line' storage, and is available even after a system reboot.

22.2.3.4 enum DDS::HistoryQosPolicyKind

This enumeration contains the kinds of History.

Enumerator

KEEP_LAST_HISTORY_QOS Store at most *depth* Samples in the data cache for each Instance. Samples may be overwritten.

KEEP_ALL_HISTORY_QOS Store Samples for each Instance (up to the limits specified by **ResourceLimitsQosPolicy** (p. 97)) until the **DDS** (p. 313) infrastructure is finished with them. Until this time, Samples may not be overwritten.

22.2.3.5 enum DDS::LivelinessQosPolicyKind

This enumeration contains the kinds of Liveliness.

Enumerator

AUTOMATIC_LIVELINESS_QOS The **DomainParticipant** (p. 44) automatically asserts liveliness for all DataWriters within the **DomainParticipant** (p. 44).

MANUAL_BY_PARTICIPANT_LIVELINESS_QOS The application manually asserts the liveliness all DataWriters within this **DomainParticipant** (p. 44) by calling assert_liveliness() or writing data on any one **DataWriter** (p. 37) within this **DomainParticipant** (p. 44).

MANUAL_BY_TOPIC_LIVELINESS_QOS The application manually asserts the liveliness for each individual **DataWriter** (p. 37) by calling assert_liveliness() or writing data on that **DataWriter** (p. 37).

22.2.3.6 enum DDS::OwnershipQosPolicyKind

This enumeration contains the kinds of Ownership.

Enumerator

SHARED_OWNERSHIP_QOS DataReaders receive data published by multiple DataWriters.

EXCLUSIVE_OWNERSHIP_QOS DataReaders receive data published by the highest strength DataWriter for each Instance.

22.2.3.7 enum DDS::PresentationQosPolicyAccessScopeKind

This enumeration contains the kinds of Access Scope for the **PresentationQosPolicy** (p. 94).

Enumerator

INSTANCE_PRESENTATION_QOS The scope is within an Instance.

TOPIC_PRESENTATION_QOS The scope is within samples published by DataWriters on the **Topic** (p. 69).

GROUP_PRESENTATION_QOS The scope is within samples published by DataWriters within the **Publisher** (p. 57).

22.2.3.8 enum DDS::ReliabilityQosPolicyKind

This enumeration contains the kinds of Reliability.

Enumerator

BEST_EFFORT_RELIABILITY_QOS Best effort delivery – NO retransmission, HB, or ACKNACK.

RELIABLE_RELIABILITY_QOS Reliable delivery – HB, ACKNACK, and re-transmission.

22.2.3.9 enum DDS::TypeConsistencyKind

the allowed kinds of TypeConsistency

Enumerator

DISALLOW_TYPE_COERCION disallow coercion (type name and structure must match)

ALLOW_TYPE_COERCION allow coercion

22.3 DDS::rpc Namespace Reference

Provides the RPC over DDS infrastructure.

Namespaces

- **example**

  Provides an example of generated RPC over DDS code. The structures are generated for this IDL:

Classes

- class **ClientEndpoint**
- class **ClientParams**

  Used to pass configuration parameters when constructing a Client.
- class **ListenerBase**

  Basis for all rpc related listener classes.
- class **Replier**

  A replier receives requests and send replies.
- class **ReplierBase**

  *Provides a generic basis for all **Replier** (p. 207) objects.*
- class **ReplierListener**

  *ReplierLister can be installed on a **Replier** (p. 207).*
- class **ReplierParams**

  *Used to pass configuration parameters when constructing a **Replier** (p. 207).*
- class **Requester**

  *A **Requester** (p. 214) sends requests and receives replies.*
- class **RequesterListener**

  ***RequesterListener** (p. 221) can be installed on a **Requester** (p. 214).*
- class **RequesterParams**

  *Used to pass configuration parameters when constructing a **Requester** (p. 214).*
- class **RPCEntity**

  ***RPCEntity** (p. 225) is a base abstract class.*
- class **ServiceEndpoint**
- class **ServiceParams**

  Used to pass configuration parameters when constructing a Service.
- class **ServiceProxy**

  ***ServiceProxy** (p. 228).*
- class **SimpleReplierListener**

  ***SimpleReplierListener** (p. 230) can be installed on a **Replier** (p. 207).*
- class **SimpleRequesterListener**

  *SimpleReqeusterListener can be installed on a **Requester** (p. 214).*

Enumerations

22.3.1 Detailed Description

Provides the RPC over DDS infrastructure.

namespace rpc

22.4 DDS::rpc::example Namespace Reference

Provides an example of generated RPC over DDS code. The structures are generated for this IDL:

Classes

- struct **Foo**
- class **Foo_Call**
- struct **Foo_op1_In**
- struct **Foo_op1_Out**
- class **Foo_op1_Result**
- struct **Foo_op2_In**
- struct **Foo_op2_Out**
- class **Foo_op2_Result**
- struct **Foo_Reply**
- struct **Foo_Request**
- class **Foo_Return**
- struct **FooClient**
- struct **FooService**

22.4.1 Detailed Description

Provides an example of generated RPC over DDS code. The structures are generated for this IDL:

namespace rpc

```
@service
 interface Foo (p.233) {
    boolean op1( long param );
    long    op2(  );
};
```

Index

~QosProvider
DDS Quality of Service, 7

ALLOW_TYPE_COERCION
DDS, 320
AUTOMATIC_LIVELINESS_QOS
DDS, 319
absolute_generation_rank
DDS_SampleInfo, 144
add_member
DDS::DynamicTypeBuilder, 174
advertise_meta_multicast
CoreDX_UdpTransportConfig, 201
advertise_user_multicast
CoreDX_UdpTransportConfig, 201
AnnotationDescriptor
DDS::AnnotationDescriptor, 149
apply_annotation
DDS::DynamicTypeBuilder, 175
assert_liveliness
DDS::DataWriter, 39
DDS::DomainParticipant, 46
at
DDS::sequence, 147
attach_condition
DDS::WaitSet, 73

BEST_EFFORT_RELIABILITY_QOS
DDS, 320
BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS
DDS, 318
BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS
DDS, 318
back
DDS::sequence, 147
base_type
DDS::TypeDescriptor, 192
begin_access
DDS::Subscriber, 66
begin_coherent_changes
DDS::Publisher, 58
bind
DDS::rpc::ServiceProxy, 229
bound
DDS::TypeDescriptor, 193
broadcast_address
CoreDX_UdpTransportConfig, 201
buffer
DDS::sequence, 147
build
DDS::DynamicTypeBuilder, 175
BuiltinTopicKey_t
DDS::BuiltinTopicKey_t, 131

CDX::ArrayDynamicType, 245
get_element, 246
get_element_type, 246
get_length, 246
set_element, 246
set_element_type, 247
set_length, 247
set_max_length, 247
CDX::BooleanDynamicType, 248
get_boolean, 248
set_boolean, 248
CDX::CharDynamicType, 249
get_char, 249
set_char, 249
CDX::DoubleDynamicType, 250
get_double, 250
set_double, 250
CDX::DynamicType, 251
get_type, 252
CDX::DynamicTypeDataReader, 252
CDX::DynamicTypeDataWriter, 252
CDX::EnumConstant, 253
name, 253
value, 253
CDX::EnumDynamicType, 253
get_constant, 254
get_constant_by_name, 254
get_constant_by_value, 254
get_num_constants, 254
get_value, 254
set_constant, 254
set_num_constants, 254
set_value, 254
CDX::FloatDynamicType, 255
get_float, 255
set_float, 255
CDX::LongDynamicType, 255
get_long, 256

set_long, 256
CDX::LongLongDynamicType, 256
get_longlong, 257
set_longlong, 257
CDX::OctetDynamicType, 257
get_octet, 258
set_octet, 258
CDX::SequenceDynamicType, 258
get_element, 259
get_element_type, 259
get_length, 260
set_element, 260
set_element_type, 260
set_length, 260
set_max_length, 261
CDX::ShortDynamicType, 261
get_short, 262
set_short, 262
CDX::StringDynamicType, 262
get_length, 263
get_max_length, 263
get_string, 263
set_max_length, 263
set_string, 264
CDX::StructDynamicType, 264
get_field, 265
get_field_key, 265
get_field_name, 265
get_num_fields, 265
set_field, 265
set_num_fields, 266
CDX::ULongDynamicType, 266
get_ulong, 267
set_ulong, 267
CDX::ULongLongDynamicType, 267
get_ulonglong, 268
set_ulonglong, 268
CDX::UShortDynamicType, 273
get_ushort, 274
set_ushort, 274
CDX::UnionDynamicType, 268
get_default_field, 269
get_discriminator, 269
get_field, 270
get_field_label, 270
get_field_name, 270
get_field_num_labels, 270
get_num_fields, 270
get_selected_field, 271
set_default_field, 271
set_discriminator, 271
set_field, 272
set_field_label, 272
set_field_num_labels, 272
set_num_fields, 273
CDX_XmlApi, 248
CENTRAL_DISCOVERY_QOS
DDS, 319
CLOSED
CoreDX DDS RPC, 17
CacheStatistics, 141
check_version
DDS::DomainParticipant, 46
clear
DDS::sequence, 147
clear_all_values
DDS::DynamicData, 155
clear_nonkey_values
DDS::DynamicData, 155
clear_value
DDS::DynamicData, 155
clone
DDS::DynamicData, 155
close
DDS::rpc::RPCEntity, 225
const_iterator
DDS::LoanedSamples, 121
DDS::SharedSamples, 128
const_value_type
DDS::LoanedSamples, 121
DDS::SharedSamples, 128
contains_entity
DDS::DomainParticipant, 46
copy_from
DDS::AnnotationDescriptor, 150
DDS::MemberDescriptor, 190
DDS::TypeDescriptor, 192
copy_from_topic_qos
DDS::Publisher, 58
DDS::Subscriber, 66
CoreDX DDS 'Sample' objects., 12
CoreDX DDS DynamicTypes, 20
CoreDX DDS RPC, 16
CLOSED, 17
PAUSED, 17
RUNNING, 17
ServiceStatus, 17
CoreDX DDS Transports, 15
CoreDX DDS XML API, 18
CoreDX_LmtTransport, 195
CoreDX_LmtTransport_clear_config, 195
CoreDX_LmtTransport_create_transport, 195
CoreDX_LmtTransport_get_default_config, 196
CoreDX_LmtTransport_get_env_config, 196
CoreDX_LmtTransport_clear_config
CoreDX_LmtTransport, 195
CoreDX_LmtTransport_create_transport
CoreDX_LmtTransport, 195

CoreDX_LmtTransport_get_default_config
CoreDX_LmtTransport, 196
CoreDX_LmtTransport_get_env_config
CoreDX_LmtTransport, 196
CoreDX_LmtTransportConfig, 196
debug_flags, 196
max_rx_buf_size, 196
max_tx_size, 197
so_rcvbuf, 197
so_sndbuf, 197
CoreDX_TcpTransport, 197
CoreDX_TcpTransport_clear_config, 197
CoreDX_TcpTransport_create_transport, 197
CoreDX_TcpTransport_get_default_config, 198
CoreDX_TcpTransport_get_env_config, 198
CoreDX_TcpTransport_clear_config
CoreDX_TcpTransport, 197
CoreDX_TcpTransport_create_transport
CoreDX_TcpTransport, 197
CoreDX_TcpTransport_get_default_config
CoreDX_TcpTransport, 198
CoreDX_TcpTransport_get_env_config
CoreDX_TcpTransport, 198
CoreDX_TcpTransportConfig, 198
debug_flags, 198
dynamic_interfaces, 198
interfaces, 199
participant_index, 199
tx_max_packet_size, 199
CoreDX_UdpTransport, 199
CoreDX_UdpTransport_clear_config, 199
CoreDX_UdpTransport_create_transport, 199
CoreDX_UdpTransport_get_default_config, 200
CoreDX_UdpTransport_get_env_config, 200
CoreDX_UdpTransport_clear_config
CoreDX_UdpTransport, 199
CoreDX_UdpTransport_create_transport
CoreDX_UdpTransport, 199
CoreDX_UdpTransport_get_default_config
CoreDX_UdpTransport, 200
CoreDX_UdpTransport_get_env_config
CoreDX_UdpTransport, 200
CoreDX_UdpTransportConfig, 200
advertise_meta_multicast, 201
advertise_user_multicast, 201
broadcast_address, 201
debug_flags, 201
do_meta_broadcast, 201
dynamic_interfaces, 201
interfaces, 201
meta_multicast_address_v4, 201
meta_multicast_address_v6, 201
multicast_ttl, 202
participant_index, 202
rx_init_buffer_size, 202
rx_max_buffer_size, 202
rx_meta_multicast, 202
rx_user_multicast, 202
so_rcvbuf, 202
so_sndbuf, 202
tx_max_packet_size, 202
tx_meta_multicast, 202
tx_meta_unicast, 202
use_ipv4, 203
use_ipv6, 203
user_multicast_address_v4, 203
user_multicast_address_v6, 203
count
DDS_QosPolicyCount, 143
cpp/coredx_lmt_transport.h, 294
cpp/coredx_tcp_transport.h, 295
cpp/coredx_udp_transport.h, 296
cpp/dds.hh, 283
cpp/dds_builtin.hh, 287
cpp/dds_dtype.hh, 289
cpp/dds_seq.hh, 291
cpp/dds_types.h, 292
cpp/function_call.hh, 297
cpp/qos_provider.hh, 297
cpp/request_reply.hh, 298
cpp/sample.hh, 299
cpp/xml_api.h, 300
cpp/xtypes.hh, 300
cpp/xtypes_dtype.hh, 301
create_alias_type
DDS::DynamicTypeBuilderFactory, 180
create_array_type
DDS::DynamicTypeBuilderFactory, 180
create_bitset_type
DDS::DynamicTypeBuilderFactory, 180
create_contentfilteredtopic
DDS::DomainParticipant, 47
create_data
DDS::DynamicDataFactory, 169
create_datareader
DDS::Subscriber, 66
create_datawriter
DDS::Publisher, 58
create_enumeration_type
DDS::DynamicTypeBuilderFactory, 180
create_map_type
DDS::DynamicTypeBuilderFactory, 181
create_multitopic
DDS::DomainParticipant, 47
create_participant
DDS::DomainParticipantFactory, 54
create_publisher
DDS::DomainParticipant, 47

create_querycondition
DDS::DataReader, 28
create_readcondition
DDS::DataReader, 29
create_sequence_type
DDS::DynamicTypeBuilderFactory, 181
create_string_type
DDS::DynamicTypeBuilderFactory, 181
create_structure_type
DDS::DynamicTypeBuilderFactory, 182
create_subscriber
DDS::DomainParticipant, 47
create_topic
DDS::DomainParticipant, 48
create_type
DDS::DynamicTypeBuilderFactory, 182
create_type_copy
DDS::DynamicTypeBuilderFactory, 182
create_type_support
DDS::DynamicTypeSupport, 189
create_type_w_document
DDS::DynamicTypeBuilderFactory, 183
create_type_w_type_object
DDS::DynamicTypeBuilderFactory, 183
create_type_w_uri
DDS::DynamicTypeBuilderFactory, 183
create_union_type
DDS::DynamicTypeBuilderFactory, 184
create_wstring_type
DDS::DynamicTypeBuilderFactory, 184

DCPSParticipant
DDS::DCPSParticipant, 137
DCPSPublication
DDS::DCPSPublication, 139
DCPSSubscription
DDS::DCPSSubscription, 140
DDS Conditions, 10
DDS Conditions, Listeners, and WaitSets, 8
DDS Entities, 3
DDS Listeners, 9
DDS Quality of Service, 4
~QosProvider, 7
get_datareader_qos, 5
get_datawriter_qos, 5
get_participant_qos, 5
get_participantfactory_qos, 5
get_publisher_qos, 5
get_subscriber_qos, 5
get_topic_qos, 6
QosProvider, 6
return_datareader_qos, 6
return_datawriter_qos, 6
return_participant_qos, 6
return_participantfactory_qos, 6
return_publisher_qos, 6
return_subscriber_qos, 6
return_topic_qos, 7
DDS Status Structures, 13
DDS WaitSets, 11
DDS::AnnotationDescriptor, 149
AnnotationDescriptor, 149
copy_from, 150
equals, 150
get_all_value, 150
get_value, 150
is_consistent, 150
set_value, 150
type, 151
DDS::BuiltinTopicKey_t, 131
BuiltinTopicKey_t, 131
value, 131
DDS::Condition, 23
get_trigger_value, 23
DDS::ContentFilteredTopic, 24
get_expression_parameters, 25
get_name, 25
get_related_topic, 25
get_type_name, 25
set_expression_parameters, 25
DDS::DCPSParticipant, 137
DCPSParticipant, 137
DDS::DCPSPublication, 137
DCPSPublication, 139
DDS::DCPSSubscription, 139
DCPSSubscription, 140
DDS::DataReader, 26
create_querycondition, 28
create_readcondition, 29
delete_contained_entities, 29
delete_readcondition, 29
enable, 29
get_key_value, 30
get_listener, 30
get_liveliness_changed_status, 30
get_matched_publication_data, 30
get_matched_publications, 30
get_qos, 31
get_requested_deadline_missed_status, 31
get_requested_incompatible_qos_status, 31
get_sample_lost_status, 31
get_sample_rejected_status, 31
get_status_changes, 31
get_subscription_matched_status, 31
get_topicdescription, 32
lookup_instance, 32
read, 32
read_instance, 33

read_next_instance, 33
read_next_instance_w_condition, 33
read_next_sample, 34
read_w_condition, 34
return_loan, 34
set_listener, 34
set_qos, 35
take, 35
take_instance, 36
take_next_instance, 36
take_next_instance_w_condition, 36
take_next_sample, 36
take_w_condition, 37
DDS::DataReaderListener, 107
on_data_available, 108
on_liveliness_changed, 108
on_requested_deadline_missed, 108
on_requested_incompatible_qos, 108
on_sample_lost, 108
on_sample_rejected, 108
on_subscription_matched, 108
DDS::DataReaderQos, 75
DDS::DataRepresentationQosPolicy, 83
DataRepresentationQosPolicy, 83
DDS::DataWriter, 37
assert_liveliness, 39
dispose, 39
dispose_w_timestamp, 39
enable, 39
get_key_value, 40
get_listener, 40
get_liveliness_lost_status, 40
get_matched_subscription_data, 40
get_matched_subscriptions, 40
get_offered_deadline_missed_status, 41
get_offered_incompatible_qos_status, 41
get_publication_matched_status, 41
get_qos, 41
get_status_changes, 41
lookup_instance, 41
register_instance, 41
register_instance_w_timestamp, 42
set_listener, 42
set_qos, 42
unregister_instance, 42
unregister_instance_w_timestamp, 42
wait_for_acknowledgments, 42
write, 43
write_w_timestamp, 43
DDS::DataWriterListener, 109
on_liveliness_lost, 109
on_offered_deadline_missed, 109
on_offered_incompatible_qos, 110
on_publication_matched, 110
DDS::DataWriterQos, 76
DDS::DeadlineQosPolicy, 83
DeadlineQosPolicy, 84
DDS::DestinationOrderQosPolicy, 84
DestinationOrderQosPolicy, 85
DDS::DomainEntity, 43
DDS::DomainParticipant, 44
assert_liveliness, 46
check_version, 46
contains_entity, 46
create_contentfilteredtopic, 47
create_multitopic, 47
create_publisher, 47
create_subscriber, 47
create_topic, 48
delete_contained_entities, 48
delete_contentfilteredtopic, 48
delete_multitopic, 48
delete_publisher, 48
delete_subscriber, 49
delete_topic, 49
do_work, 49
enable, 49
find_topic, 50
get_builtin_subscriber, 50
get_default_publisher_qos, 50
get_default_subscriber_qos, 50
get_default_topic_qos, 50
get_discovered_participant_data, 50
get_discovered_participants, 51
get_discovered_topic_data, 51
get_discovered_topics, 51
get_listener, 51
get_status_changes, 51
ignore_participant, 52
ignore_publication, 52
ignore_subscription, 52
ignore_topic, 52
lookup_topicdescription, 52
register_type, 52
set_default_publisher_qos, 53
set_default_subscriber_qos, 53
set_default_topic_qos, 53
set_listener, 53
set_qos, 53
DDS::DomainParticipantFactory, 54
create_participant, 54
delete_participant, 54
get_default_participant_qos, 55
get_instance, 55
lookup_participant, 55
set_default_participant_qos, 55
set_qos, 55
DDS::DomainParticipantFactoryQos, 77

DDS::DomainParticipantListener, 110
DDS::DomainParticipantQos, 78
DDS::DurabilityQosPolicy, 85
DurabilityQosPolicy, 86
DDS::DurabilityServiceQosPolicy, 86
DurabilityServiceQosPolicy, 87
DDS::Duration_t, 131
Duration_t, 132
DDS::DynamicData, 151
clear_all_values, 155
clear_nonkey_values, 155
clear_value, 155
clone, 155
DynamicData, 154
equals, 155
get_boolean_value, 156
get_boolean_values, 156
get_byte_value, 156
get_byte_values, 156
get_char32_value, 157
get_char32_values, 157
get_char8_value, 157
get_char8_values, 157
get_complex_value, 157
get_descriptor, 158
get_float32_value, 158
get_float32_values, 158
get_float64_value, 158
get_float64_values, 158
get_int16_value, 158
get_int16_values, 159
get_int32_value, 159
get_int32_values, 159
get_int64_value, 159
get_int64_values, 159
get_item_count, 160
get_string_value, 160
get_string_values, 160
get_uint16_value, 160
get_uint16_values, 161
get_uint32_value, 161
get_uint32_values, 161
get_uint64_value, 161
get_uint64_values, 161
get_wstring_value, 161
get_wstring_values, 162
loan_value, 162
return_loaned_value, 162
set_boolean_value, 162
set_boolean_values, 162
set_byte_value, 163
set_byte_values, 163
set_char32_value, 163
set_char32_values, 163
set_char8_value, 163
set_char8_values, 164
set_complex_value, 164
set_float32_value, 164
set_float32_values, 164
set_float64_value, 164
set_float64_values, 165
set_int16_value, 165
set_int16_values, 165
set_int32_value, 165
set_int32_values, 165
set_int64_value, 165
set_int64_values, 166
set_string_value, 166
set_string_values, 166
set_uint16_value, 166
set_uint16_values, 166
set_uint32_value, 167
set_uint32_values, 167
set_uint64_value, 167
set_uint64_values, 167
set_wstring_value, 167
set_wstring_values, 168
DDS::DynamicDataFactory, 168
create_data, 169
DynamicDataFactory, 168
DDS::DynamicDataReader, 169
DDS::DynamicDataWriter, 169
DDS::DynamicType, 170
DynamicType, 171
equals, 171
get_all_members, 171
get_all_members_by_name, 171
get_annotation, 171
get_annotation_count, 172
get_descriptor, 172
get_kind, 172
get_member, 172
get_member_by_name, 173
get_name, 173
DDS::DynamicTypeBuilder, 173
add_member, 174
apply_annotation, 175
build, 175
delete_type, 175
DynamicTypeBuilder, 174
equals, 175
get_all_members, 176
get_all_members_by_name, 176
get_annotation, 176
get_annotation_count, 176
get_descriptor, 176
get_kind, 177
get_member, 177

get_member_by_name, 177
get_name, 178
DDS::DynamicTypeBuilderFactory, 178
create_alias_type, 180
create_array_type, 180
create_bitset_type, 180
create_enumeration_type, 180
create_map_type, 181
create_sequence_type, 181
create_string_type, 181
create_structure_type, 182
create_type, 182
create_type_copy, 182
create_type_w_document, 183
create_type_w_type_object, 183
create_type_w_uri, 183
create_union_type, 184
create_wstring_type, 184
delete_instance, 184
delete_type, 185
delete_type_builder, 185
DynamicTypeBuilderFactory, 179
get_instance, 185
get_primitive_type, 185
DDS::DynamicTypeMember, 186
DynamicTypeMember, 187
equals, 187
get_annotation, 187
get_annotation_count, 187
get_descriptor, 187
get_id, 187
get_name, 188
DDS::DynamicTypeSupport, 188
create_type_support, 189
delete_type_support, 189
DynamicTypeSupport, 189
get_type, 189
get_type_name, 189
DDS::Entity, 55
enable, 56
DDS::GUID_t, 132
GUID_t, 132
value, 132
DDS::GroupDataQosPolicy, 88
GroupDataQosPolicy, 88
DDS::GuardCondition, 56
get_trigger_value, 57
set_trigger_value, 57
DDS::HistoryQosPolicy, 88
HistoryQosPolicy, 89
DDS::LatencyBudgetQosPolicy, 89
LatencyBudgetQosPolicy, 90
DDS::LifespanQosPolicy, 90
LifespanQosPolicy, 90
DDS::LivelinessQosPolicy, 90
LivelinessQosPolicy, 91
DDS::LoanedSamples
const_iterator, 121
const_value_type, 121
difference_type, 121
iterator, 121
move_construct_from_loans, 121
release, 121
return_loan, 121
to_shared, 122
value_type, 121
DDS::LoanedSamples< T >, 119
DDS::MemberDescriptor, 190
copy_from, 190
equals, 190
is_consistent, 190
MemberDescriptor, 190
DDS::OwnershipQosPolicy, 92
OwnershipQosPolicy, 92
DDS::OwnershipStrengthQosPolicy, 92
OwnershipStrengthQosPolicy, 93
DDS::PartitionQosPolicy, 93
PartitionQosPolicy, 94
DDS::PresentationQosPolicy, 94
PresentationQosPolicy, 95
DDS::Property_t, 132
Property_t, 133
DDS::PropertyQosPolicy, 95
PropertyQosPolicy, 95
DDS::Publisher, 57
begin_coherent_changes, 58
copy_from_topic_qos, 58
create_datawriter, 58
delete_contained_entities, 59
delete_datawriter, 59
enable, 59
end_coherent_changes, 59
get_default_datawriter_qos, 60
get_listener, 60
get_qos, 60
get_status_changes, 60
lookup_datawriter, 60
resume_publications, 60
set_default_datawriter_qos, 61
set_listener, 61
set_qos, 61
suspend_publications, 61
wait_for_acknowledgments, 61
DDS::PublisherListener, 110
DDS::PublisherQos, 78
partition, 79
presentation, 79
DDS::QosProvider, 105

DDS::QueryCondition, 61
get_query_expression, 62
get_query_parameters, 62
set_query_parameters, 62
DDS::ReadCondition, 63
get_trigger_value, 64
DDS::ReliabilityQosPolicy, 96
ReliabilityQosPolicy, 96
DDS::ResourceLimitsQosPolicy, 97
ResourceLimitsQosPolicy, 97
DDS::RpcQosPolicy, 97
RpcQosPolicy, 98
DDS::Sample< T >, 122
DDS::SampleIdentity_t, 133
guid, 134
SampleIdentity_t, 133
DDS::SampleIterator
difference_type, 125
value_type, 125
DDS::SampleIterator< T, IsConst >, 123
DDS::SampleRef< T >, 125
DDS::SequenceNumber_t, 134
high, 134
low, 134
SequenceNumber_t, 134
DDS::SharedSamples
const_iterator, 128
const_value_type, 128
difference_type, 128
iterator, 128
value_type, 128
DDS::SharedSamples< T >, 126
DDS::StatusCondition, 64
get_enabled_statuses, 64
get_trigger_value, 64
set_enabled_statuses, 65
DDS::Subscriber, 65
begin_access, 66
copy_from_topic_qos, 66
create_datareader, 66
delete_contained_entities, 67
delete_datareader, 67
enable, 67
get_datareaders, 68
get_default_datareader_qos, 68
get_listener, 68
get_qos, 68
get_status_changes, 68
lookup_datareader, 69
set_default_datareader_qos, 69
set_listener, 69
set_qos, 69
DDS::SubscriberListener, 111
on_data_on_readers, 112
DDS::SubscriberQos, 79
partition, 80
presentation, 80
DDS::Time_t, 134
DDS::TimeBasedFilterQosPolicy, 99
TimeBasedFilterQosPolicy, 100
DDS::Topic, 69
enable, 70
get_inconsistent_topic_status, 71
get_listener, 71
get_qos, 71
get_status_changes, 71
set_listener, 71
set_qos, 72
DDS::TopicDataQosPolicy, 100
TopicDataQosPolicy, 101
DDS::TopicDescription, 72
get_name, 72
get_type_name, 72
DDS::TopicListener, 112
on_inconsistent_topic, 112
DDS::TopicQos, 80
DDS::TransportPriorityQosPolicy, 101
TransportPriorityQosPolicy, 101
DDS::TypeConsistencyEnforcementQosPolicy, 102
TypeConsistencyEnforcementQosPolicy, 102
DDS::TypeDescriptor, 191
base_type, 192
bound, 193
copy_from, 192
element_type, 193
equals, 192
is_consistent, 192
key_element_type, 193
TypeDescriptor, 192
DDS::TypecodeQosPolicy, 101
TypecodeQosPolicy, 102
DDS::UserDataQosPolicy, 102
UserDataQosPolicy, 103
DDS::WaitSet, 73
attach_condition, 73
detach_condition, 73
get_conditions, 73
wait, 74
DDS::WriteSample< T >, 128
DDS::WriteSampleRef< T >, 129
DDS::rpc, 320
DDS::rpc::ClientEndpoint, 205
DDS::rpc::ClientParams, 206
DDS::rpc::ListenerBase, 207
DDS::rpc::RPCEntity, 225
close, 225
is_null, 225
DDS::rpc::Replier

read_request, 209
read_requests, 209
receive_nondata_samples, 209
receive_request, 210
receive_requests, 210
Replier, 209
take_request, 210
take_requests, 211
wait_for_requests, 211
DDS::rpc::Replier< TReq, TRep >, 207
DDS::rpc::ReplierBase, 211
DDS::rpc::ReplierListener
on_request_available, 212
DDS::rpc::ReplierListener< TReq, TRep >, 212
DDS::rpc::ReplierParams, 212
DDS::rpc::Requester
read_replies, 216
read_reply, 217
receive_nondata_samples, 217
receive_replies, 217, 218
receive_reply, 218, 219
Requester, 216
take_replies, 219
take_reply, 219, 220
wait_for_replies, 220, 221
DDS::rpc::Requester< TReq, TRep >, 214
DDS::rpc::RequesterListener
on_reply_available, 222
DDS::rpc::RequesterListener< TReq, TRep >, 221
DDS::rpc::RequesterParams, 222
datareader_qos, 223
datawriter_qos, 223
domain_participant, 223
publisher, 223
reply_topic_name, 224
request_topic_name, 224
requester_listener, 224
service_name, 224
simple_requester_listener, 224
subscriber, 224
DDS::rpc::ServiceEndpoint, 225
pause, 226
resume, 226
DDS::rpc::ServiceParams, 227
DDS::rpc::ServiceProxy, 228
bind, 229
get_discoverd_service_instances, 229
get_discovered_service_count, 229
get_discovered_service_instances, 229
unbind, 229
wait_for_service, 229, 230
wait_for_services, 230
DDS::rpc::SimpleReplierListener
process_request, 231
DDS::rpc::SimpleReplierListener< TReq, TRep >, 230
DDS::rpc::SimpleRequesterListener
process_reply, 232
DDS::rpc::SimpleRequesterListener< TRep >, 231
DDS::rpc::example, 321
DDS::rpc::example::Foo, 233
Foo, 233
op1, 234
op2, 234
DDS::rpc::example::Foo_Call, 234
op1, 234
op2, 234
DDS::rpc::example::Foo_Reply, 239
data, 239
Foo_Reply, 239
header, 239
related_id, 239
DDS::rpc::example::Foo_Request, 239
data, 240
Foo_Request, 240
header, 240
request_id, 240
DDS::rpc::example::Foo_Return, 240
op1, 241
op2, 241
unknownOp, 241
DDS::rpc::example::Foo_op1_In, 235
Foo_op1_In, 235
param, 235
DDS::rpc::example::Foo_op1_Out, 235
Foo_op1_Out, 236
return_, 236
DDS::rpc::example::Foo_op1_Result, 236
result, 236
unknownEx, 236
DDS::rpc::example::Foo_op2_In, 237
dummy, 237
Foo_op2_In, 237
DDS::rpc::example::Foo_op2_Out, 237
Foo_op2_Out, 238
return_, 238
DDS::rpc::example::Foo_op2_Result, 238
result, 238
unknownEx, 238
DDS::rpc::example::FooClient, 241
FooClient, 242
op1, 242
op2, 242
DDS::rpc::example::FooService, 242
FooService, 243
op1, 243
op2, 243
DDS::sequence
at, 147

back, 147
buffer, 147
clear, 147
front, 147
operator[], 147, 148
pop_back, 148
push_back, 148
reserve, 148
resize, 148
swap, 148
DDS::sequence< T >, 145
DDS_CacheStatistics
instance_alive_count, 141
instance_count, 142
instance_disposed_count, 142
instance_new_count, 142
instance_no_writers_count, 142
sample_count, 142
sample_read_count, 142
state_flags, 142
DDS_DomainId_t
dds_types.h, 293
DDS_InstanceHandle_t
dds_types.h, 293
DDS_QosPolicyCount
count, 143
policy_id, 143
DDS_ReturnCode_t
dds_types.h, 293
DDS_SampleInfo
absolute_generation_rank, 144
generation_rank, 144
instance_state, 145
sample_state, 145
valid_data, 145
view_state, 145
DDS_StatusMask
dds_types.h, 293
DDS_Time_t, 135
nanosec, 135
sec, 135
DDS, 313
ALLOW_TYPE_COERCION, 320
AUTOMATIC_LIVELINESS_QOS, 319
BEST_EFFORT_RELIABILITY_QOS, 320
BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS, 318
BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS, 318
CENTRAL_DISCOVERY_QOS, 319
DISALLOW_TYPE_COERCION, 320
DestinationOrderQosPolicyKind, 318
DiscoveryQosPolicyDiscoveryKind, 318
DomainId_t, 318
DurabilityQosPolicyKind, 319
EXCLUSIVE_OWNERSHIP_QOS, 320
GROUP_PRESENTATION_QOS, 320
GuidPrefix_t, 318
HistoryQosPolicyKind, 319
INSTANCE_PRESENTATION_QOS, 320
InstanceHandle_t, 318
KEEP_ALL_HISTORY_QOS, 319
KEEP_LAST_HISTORY_QOS, 319
LivelinessQosPolicyKind, 319
MANUAL_BY_PARTICIPANT_LIVELINESS_QOS, 319
MANUAL_BY_TOPIC_LIVELINESS_QOS, 319
OwnershipQosPolicyKind, 319
PEER_DISCOVERY_QOS, 319
PERSISTENT_DURABILITY_QOS, 319
PresentationQosPolicyAccessScopeKind, 320
RELIABLE_RELIABILITY_QOS, 320
ReliabilityQosPolicyKind, 320
ReturnCode_t, 318
SHARED_OWNERSHIP_QOS, 320
StatusMask, 318
TOPIC_PRESENTATION_QOS, 320
TRANSIENT_DURABILITY_QOS, 319
TRANSIENT_LOCAL_DURABILITY_QOS, 319
TypeConsistencyKind, 320
VOLATILE_DURABILITY_QOS, 319
DISALLOW_TYPE_COERCION
DDS, 320
data
DDS::rpc::example::Foo_Reply, 239
DDS::rpc::example::Foo_Request, 240
DataRepresentationQosPolicy
DDS::DataRepresentationQosPolicy, 83
datareader_qos
DDS::rpc::RequesterParams, 223
datawriter_qos
DDS::rpc::RequesterParams, 223
dds_dtype.hh
TYPECODE_ALIAS, 290
TYPECODE_ARRAY, 290
TYPECODE_BOOLEAN, 290
TYPECODE_CHAR, 290
TYPECODE_DOUBLE, 290
TYPECODE_ENUM, 290
TYPECODE_FLOAT, 290
TYPECODE_LONGDOUBLE, 290
TYPECODE_LONGLONG, 290
TYPECODE_LONG, 290
TYPECODE_OCTET, 290
TYPECODE_SEQUENCE, 290
TYPECODE_SHORT, 290
TYPECODE_STRING, 290
TYPECODE_STRUCT, 290

TYPECODE_ULONGLONG, 290
TYPECODE_ULONG, 290
TYPECODE_UNION, 290
TYPECODE_USHORT, 290
TYPECODE_VALUE_PARAM, 290
TYPECODE_VALUE, 290
TYPECODE_WCHAR, 290
TYPECODE_WSTRING, 290
TypeCodeKind, 290
dds_types.h
DDS_DomainId_t, 293
DDS_InstanceHandle_t, 293
DDS_ReturnCode_t, 293
DDS_StatusMask, 293
DeadlineQosPolicy
DDS::DeadlineQosPolicy, 84
debug_flags
CoreDX_LmtTransportConfig, 196
CoreDX_TcpTransportConfig, 198
CoreDX_UdpTransportConfig, 201
delete_contained_entities
DDS::DataReader, 29
DDS::DomainParticipant, 48
DDS::Publisher, 59
DDS::Subscriber, 67
delete_contentfilteredtopic
DDS::DomainParticipant, 48
delete_datareader
DDS::Subscriber, 67
delete_datawriter
DDS::Publisher, 59
delete_instance
DDS::DynamicTypeBuilderFactory, 184
delete_multitopic
DDS::DomainParticipant, 48
delete_participant
DDS::DomainParticipantFactory, 54
delete_publisher
DDS::DomainParticipant, 48
delete_readcondition
DDS::DataReader, 29
delete_subscriber
DDS::DomainParticipant, 49
delete_topic
DDS::DomainParticipant, 49
delete_type
DDS::DynamicTypeBuilder, 175
DDS::DynamicTypeBuilderFactory, 185
delete_type_builder
DDS::DynamicTypeBuilderFactory, 185
delete_type_support
DDS::DynamicTypeSupport, 189
DestinationOrderQosPolicy
DDS::DestinationOrderQosPolicy, 85
DestinationOrderQosPolicyKind
DDS, 318
detach_condition
DDS::WaitSet, 73
difference_type
DDS::LoanedSamples, 121
DDS::SampleIterator, 125
DDS::SharedSamples, 128
DiscoveryQosPolicy, 85
DiscoveryQosPolicyDiscoveryKind
DDS, 318
dispose
DDS::DataWriter, 39
dispose_w_timestamp
DDS::DataWriter, 39
do_meta_broadcast
CoreDX_UdpTransportConfig, 201
do_work
DDS::DomainParticipant, 49
domain_participant
DDS::rpc::RequesterParams, 223
DomainId_t
DDS, 318
dummy
DDS::rpc::example::Foo_op2_In, 237
DurabilityQosPolicy
DDS::DurabilityQosPolicy, 86
DurabilityQosPolicyKind
DDS, 319
DurabilityServiceQosPolicy
DDS::DurabilityServiceQosPolicy, 87
Duration_t
DDS::Duration_t, 132
dynamic_interfaces
CoreDX_TcpTransportConfig, 198
CoreDX_UdpTransportConfig, 201
DynamicData
DDS::DynamicData, 154
DynamicDataFactory
DDS::DynamicDataFactory, 168
DynamicType
DDS::DynamicType, 171
DynamicTypeBuilder
DDS::DynamicTypeBuilder, 174
DynamicTypeBuilderFactory
DDS::DynamicTypeBuilderFactory, 179
DynamicTypeMember
DDS::DynamicTypeMember, 187
DynamicTypeSupport
DDS::DynamicTypeSupport, 189

EXCLUSIVE_OWNERSHIP_QOS
DDS, 320
element_type

DDS::TypeDescriptor, 193
enable
DDS::DataReader, 29
DDS::DataWriter, 39
DDS::DomainParticipant, 49
DDS::Entity, 56
DDS::Publisher, 59
DDS::Subscriber, 67
DDS::Topic, 70
end_coherent_changes
DDS::Publisher, 59
EntityFactoryQosPolicy, 87
EntityNameQosPolicy, 88
equals
DDS::AnnotationDescriptor, 150
DDS::DynamicData, 155
DDS::DynamicType, 171
DDS::DynamicTypeBuilder, 175
DDS::DynamicTypeMember, 187
DDS::MemberDescriptor, 190
DDS::TypeDescriptor, 192

find_topic
DDS::DomainParticipant, 50
Foo
DDS::rpc::example::Foo, 233
Foo_Reply
DDS::rpc::example::Foo_Reply, 239
Foo_Request
DDS::rpc::example::Foo_Request, 240
Foo_op1_In
DDS::rpc::example::Foo_op1_In, 235
Foo_op1_Out
DDS::rpc::example::Foo_op1_Out, 236
Foo_op2_In
DDS::rpc::example::Foo_op2_In, 237
Foo_op2_Out
DDS::rpc::example::Foo_op2_Out, 238
FooClient
DDS::rpc::example::FooClient, 242
FooService
DDS::rpc::example::FooService, 243
front
DDS::sequence, 147

GROUP_PRESENTATION_QOS
DDS, 320
GUID_t
DDS::GUID_t, 132
generation_rank
DDS_SampleInfo, 144
get_all_members
DDS::DynamicType, 171
DDS::DynamicTypeBuilder, 176
get_all_members_by_name
DDS::DynamicType, 171
DDS::DynamicTypeBuilder, 176
get_all_value
DDS::AnnotationDescriptor, 150
get_annotation
DDS::DynamicType, 171
DDS::DynamicTypeBuilder, 176
DDS::DynamicTypeMember, 187
get_annotation_count
DDS::DynamicType, 172
DDS::DynamicTypeBuilder, 176
DDS::DynamicTypeMember, 187
get_boolean
CDX::BooleanDynamicType, 248
get_boolean_value
DDS::DynamicData, 156
get_boolean_values
DDS::DynamicData, 156
get_builtin_subscriber
DDS::DomainParticipant, 50
get_byte_value
DDS::DynamicData, 156
get_byte_values
DDS::DynamicData, 156
get_char
CDX::CharDynamicType, 249
get_char32_value
DDS::DynamicData, 157
get_char32_values
DDS::DynamicData, 157
get_char8_value
DDS::DynamicData, 157
get_char8_values
DDS::DynamicData, 157
get_complex_value
DDS::DynamicData, 157
get_conditions
DDS::WaitSet, 73
get_constant
CDX::EnumDynamicType, 254
get_constant_by_name
CDX::EnumDynamicType, 254
get_constant_by_value
CDX::EnumDynamicType, 254
get_datareader_qos
DDS Quality of Service, 5
get_datareaders
DDS::Subscriber, 68
get_datawriter_qos
DDS Quality of Service, 5
get_default_datareader_qos
DDS::Subscriber, 68
get_default_datawriter_qos
DDS::Publisher, 60

get_default_field
CDX::UnionDynamicType, 269
get_default_participant_qos
DDS::DomainParticipantFactory, 55
get_default_publisher_qos
DDS::DomainParticipant, 50
get_default_subscriber_qos
DDS::DomainParticipant, 50
get_default_topic_qos
DDS::DomainParticipant, 50
get_descriptor
DDS::DynamicData, 158
DDS::DynamicType, 172
DDS::DynamicTypeBuilder, 176
DDS::DynamicTypeMember, 187
get_discoverd_service_instances
DDS::rpc::ServiceProxy, 229
get_discovered_participant_data
DDS::DomainParticipant, 50
get_discovered_participants
DDS::DomainParticipant, 51
get_discovered_service_count
DDS::rpc::ServiceProxy, 229
get_discovered_service_instances
DDS::rpc::ServiceProxy, 229
get_discovered_topic_data
DDS::DomainParticipant, 51
get_discovered_topics
DDS::DomainParticipant, 51
get_discriminator
CDX::UnionDynamicType, 269
get_double
CDX::DoubleDynamicType, 250
get_element
CDX::ArrayDynamicType, 246
CDX::SequenceDynamicType, 259
get_element_type
CDX::ArrayDynamicType, 246
CDX::SequenceDynamicType, 259
get_enabled_statuses
DDS::StatusCondition, 64
get_expression_parameters
DDS::ContentFilteredTopic, 25
get_field
CDX::StructDynamicType, 265
CDX::UnionDynamicType, 270
get_field_key
CDX::StructDynamicType, 265
get_field_label
CDX::UnionDynamicType, 270
get_field_name
CDX::StructDynamicType, 265
CDX::UnionDynamicType, 270
get_field_num_labels
CDX::UnionDynamicType, 270
get_float
CDX::FloatDynamicType, 255
get_float32_value
DDS::DynamicData, 158
get_float32_values
DDS::DynamicData, 158
get_float64_value
DDS::DynamicData, 158
get_float64_values
DDS::DynamicData, 158
get_id
DDS::DynamicTypeMember, 187
get_inconsistent_topic_status
DDS::Topic, 71
get_instance
DDS::DomainParticipantFactory, 55
DDS::DynamicTypeBuilderFactory, 185
get_int16_value
DDS::DynamicData, 158
get_int16_values
DDS::DynamicData, 159
get_int32_value
DDS::DynamicData, 159
get_int32_values
DDS::DynamicData, 159
get_int64_value
DDS::DynamicData, 159
get_int64_values
DDS::DynamicData, 159
get_item_count
DDS::DynamicData, 160
get_key_value
DDS::DataReader, 30
DDS::DataWriter, 40
get_kind
DDS::DynamicType, 172
DDS::DynamicTypeBuilder, 177
get_length
CDX::ArrayDynamicType, 246
CDX::SequenceDynamicType, 260
CDX::StringDynamicType, 263
get_listener
DDS::DataReader, 30
DDS::DataWriter, 40
DDS::DomainParticipant, 51
DDS::Publisher, 60
DDS::Subscriber, 68
DDS::Topic, 71
get_liveliness_changed_status
DDS::DataReader, 30
get_liveliness_lost_status
DDS::DataWriter, 40
get_long

CDX::LongDynamicType, 256
get_longlong
CDX::LongLongDynamicType, 257
get_matched_publication_data
DDS::DataReader, 30
get_matched_publications
DDS::DataReader, 30
get_matched_subscription_data
DDS::DataWriter, 40
get_matched_subscriptions
DDS::DataWriter, 40
get_max_length
CDX::StringDynamicType, 263
get_member
DDS::DynamicType, 172
DDS::DynamicTypeBuilder, 177
get_member_by_name
DDS::DynamicType, 173
DDS::DynamicTypeBuilder, 177
get_name
DDS::ContentFilteredTopic, 25
DDS::DynamicType, 173
DDS::DynamicTypeBuilder, 178
DDS::DynamicTypeMember, 188
DDS::TopicDescription, 72
get_num_constants
CDX::EnumDynamicType, 254
get_num_fields
CDX::StructDynamicType, 265
CDX::UnionDynamicType, 270
get_octet
CDX::OctetDynamicType, 258
get_offered_deadline_missed_status
DDS::DataWriter, 41
get_offered_incompatible_qos_status
DDS::DataWriter, 41
get_participant_qos
DDS Quality of Service, 5
get_participantfactory_qos
DDS Quality of Service, 5
get_primitive_type
DDS::DynamicTypeBuilderFactory, 185
get_publication_matched_status
DDS::DataWriter, 41
get_publisher_qos
DDS Quality of Service, 5
get_qos
DDS::DataReader, 31
DDS::DataWriter, 41
DDS::Publisher, 60
DDS::Subscriber, 68
DDS::Topic, 71
get_query_expression
DDS::QueryCondition, 62
get_query_parameters
DDS::QueryCondition, 62
get_related_topic
DDS::ContentFilteredTopic, 25
get_requested_deadline_missed_status
DDS::DataReader, 31
get_requested_incompatible_qos_status
DDS::DataReader, 31
get_sample_lost_status
DDS::DataReader, 31
get_sample_rejected_status
DDS::DataReader, 31
get_selected_field
CDX::UnionDynamicType, 271
get_short
CDX::ShortDynamicType, 262
get_status_changes
DDS::DataReader, 31
DDS::DataWriter, 41
DDS::DomainParticipant, 51
DDS::Publisher, 60
DDS::Subscriber, 68
DDS::Topic, 71
get_string
CDX::StringDynamicType, 263
get_string_value
DDS::DynamicData, 160
get_string_values
DDS::DynamicData, 160
get_subscriber_qos
DDS Quality of Service, 5
get_subscription_matched_status
DDS::DataReader, 31
get_topic_qos
DDS Quality of Service, 6
get_topicdescription
DDS::DataReader, 32
get_trigger_value
DDS::Condition, 23
DDS::GuardCondition, 57
DDS::ReadCondition, 64
DDS::StatusCondition, 64
get_type
CDX::DynamicType, 252
DDS::DynamicTypeSupport, 189
get_type_name
DDS::ContentFilteredTopic, 25
DDS::DynamicTypeSupport, 189
DDS::TopicDescription, 72
get_uint16_value
DDS::DynamicData, 160
get_uint16_values
DDS::DynamicData, 161
get_uint32_value

DDS::DynamicData, 161
get_uint32_values
DDS::DynamicData, 161
get_uint64_value
DDS::DynamicData, 161
get_uint64_values
DDS::DynamicData, 161
get_ulong
CDX::ULongDynamicType, 267
get_ulonglong
CDX::ULongLongDynamicType, 268
get_ushort
CDX::UShortDynamicType, 274
get_value
CDX::EnumDynamicType, 254
DDS::AnnotationDescriptor, 150
get_wstring_value
DDS::DynamicData, 161
get_wstring_values
DDS::DynamicData, 162
GroupDataQosPolicy
DDS::GroupDataQosPolicy, 88
guid
DDS::SampleIdentity_t, 134
GuidPrefix_t
DDS, 318

header
DDS::rpc::example::Foo_Reply, 239
DDS::rpc::example::Foo_Request, 240
high
DDS::SequenceNumber_t, 134
HistoryQosPolicy
DDS::HistoryQosPolicy, 89
HistoryQosPolicyKind
DDS, 319

INSTANCE_PRESENTATION_QOS
DDS, 320
ignore_participant
DDS::DomainParticipant, 52
ignore_publication
DDS::DomainParticipant, 52
ignore_subscription
DDS::DomainParticipant, 52
ignore_topic
DDS::DomainParticipant, 52
InconsistentTopicStatus, 113
instance_alive_count
DDS_CacheStatistics, 141
instance_count
DDS_CacheStatistics, 142
instance_disposed_count
DDS_CacheStatistics, 142
instance_new_count
DDS_CacheStatistics, 142
instance_no_writers_count
DDS_CacheStatistics, 142
instance_state
DDS_SampleInfo, 145
InstanceHandle_t
DDS, 318
interfaces
CoreDX_TcpTransportConfig, 199
CoreDX_UdpTransportConfig, 201
is_consistent
DDS::AnnotationDescriptor, 150
DDS::MemberDescriptor, 190
DDS::TypeDescriptor, 192
is_null
DDS::rpc::RPCEntity, 225
iterator
DDS::LoanedSamples, 121
DDS::SharedSamples, 128

KEEP_ALL_HISTORY_QOS
DDS, 319
KEEP_LAST_HISTORY_QOS
DDS, 319
key_element_type
DDS::TypeDescriptor, 193

LatencyBudgetQosPolicy
DDS::LatencyBudgetQosPolicy, 90
LifespanQosPolicy
DDS::LifespanQosPolicy, 90
LivelinessChangedStatus, 113
LivelinessLostStatus, 114
LivelinessQosPolicy
DDS::LivelinessQosPolicy, 91
LivelinessQosPolicyKind
DDS, 319
loan_value
DDS::DynamicData, 162
Locator, 142
LoggingQosPolicy, 91
lookup_datareader
DDS::Subscriber, 69
lookup_datawriter
DDS::Publisher, 60
lookup_instance
DDS::DataReader, 32
DDS::DataWriter, 41
lookup_participant
DDS::DomainParticipantFactory, 55
lookup_topicdescription
DDS::DomainParticipant, 52
low
DDS::SequenceNumber_t, 134

MANUAL_BY_PARTICIPANT_LIVELINESS_QOS
DDS, 319
MANUAL_BY_TOPIC_LIVELINESS_QOS
DDS, 319
max_rx_buf_size
CoreDX_LmtTransportConfig, 196
max_tx_size
CoreDX_LmtTransportConfig, 197
MemberDescriptor
DDS::MemberDescriptor, 190
meta_multicast_address_v4
CoreDX_UdpTransportConfig, 201
meta_multicast_address_v6
CoreDX_UdpTransportConfig, 201
move_construct_from_loans
DDS::LoanedSamples, 121
multicast_ttl
CoreDX_UdpTransportConfig, 202

name
CDX::EnumConstant, 253
nanosec
DDS_Time_t, 135

OfferedDeadlineMissedStatus, 114
OfferedIncompatibleQosStatus, 115
on_data_available
DDS::DataReaderListener, 108
on_data_on_readers
DDS::SubscriberListener, 112
on_inconsistent_topic
DDS::TopicListener, 112
on_liveliness_changed
DDS::DataReaderListener, 108
on_liveliness_lost
DDS::DataWriterListener, 109
on_offered_deadline_missed
DDS::DataWriterListener, 109
on_offered_incompatible_qos
DDS::DataWriterListener, 110
on_publication_matched
DDS::DataWriterListener, 110
on_reply_available
DDS::rpc::RequesterListener, 222
on_request_available
DDS::rpc::ReplierListener, 212
on_requested_deadline_missed
DDS::DataReaderListener, 108
on_requested_incompatible_qos
DDS::DataReaderListener, 108
on_sample_lost
DDS::DataReaderListener, 108
on_sample_rejected
DDS::DataReaderListener, 108
on_subscription_matched
DDS::DataReaderListener, 108
op1
DDS::rpc::example::Foo, 234
DDS::rpc::example::Foo_Call, 234
DDS::rpc::example::Foo_Return, 241
DDS::rpc::example::FooClient, 242
DDS::rpc::example::FooService, 243
op2
DDS::rpc::example::Foo, 234
DDS::rpc::example::Foo_Call, 234
DDS::rpc::example::Foo_Return, 241
DDS::rpc::example::FooClient, 242
DDS::rpc::example::FooService, 243
operator[]
DDS::sequence, 147, 148
OwnershipQosPolicy
DDS::OwnershipQosPolicy, 92
OwnershipQosPolicyKind
DDS, 319
OwnershipStrengthQosPolicy
DDS::OwnershipStrengthQosPolicy, 93

PAUSED
CoreDX DDS RPC, 17
PEER_DISCOVERY_QOS
DDS, 319
PERSISTENT_DURABILITY_QOS
DDS, 319
param
DDS::rpc::example::Foo_op1_In, 235
participant_index
CoreDX_TcpTransportConfig, 199
CoreDX_UdpTransportConfig, 202
ParticipantLocator, 142
partition
DDS::PublisherQos, 79
DDS::SubscriberQos, 80
PartitionQosPolicy
DDS::PartitionQosPolicy, 94
pause
DDS::rpc::ServiceEndpoint, 226
PeerParticipantQosPolicy, 94
strict_match, 94
policy_id
DDS_QosPolicyCount, 143
pop_back
DDS::sequence, 148
presentation
DDS::PublisherQos, 79
DDS::SubscriberQos, 80
PresentationQosPolicy
DDS::PresentationQosPolicy, 95
PresentationQosPolicyAccessScopeKind
DDS, 320

process_reply
DDS::rpc::SimpleRequesterListener, 232
process_request
DDS::rpc::SimpleReplierListener, 231
Property_t
DDS::Property_t, 133
PropertyQosPolicy
DDS::PropertyQosPolicy, 95
PublicationMatchedStatus, 115
publisher
DDS::rpc::RequesterParams, 223
push_back
DDS::sequence, 148

QosPolicyCount, 143
QosProvider
DDS Quality of Service, 6

RELIABLE_RELIABILITY_QOS
DDS, 320
RTPSReaderQosPolicy, 98
RTPSWriterQosPolicy, 98
RUNNING
CoreDX DDS RPC, 17
read
DDS::DataReader, 32
read_instance
DDS::DataReader, 33
read_next_instance
DDS::DataReader, 33
read_next_instance_w_condition
DDS::DataReader, 33
read_next_sample
DDS::DataReader, 34
read_replies
DDS::rpc::Requester, 216
read_reply
DDS::rpc::Requester, 217
read_request
DDS::rpc::Replier, 209
read_requests
DDS::rpc::Replier, 209
read_w_condition
DDS::DataReader, 34
ReaderDataLifecycleQosPolicy, 95
receive_nondata_samples
DDS::rpc::Replier, 209
DDS::rpc::Requester, 217
receive_replies
DDS::rpc::Requester, 217, 218
receive_reply
DDS::rpc::Requester, 218, 219
receive_request
DDS::rpc::Replier, 210
receive_requests
DDS::rpc::Replier, 210
register_instance
DDS::DataWriter, 41
register_instance_w_timestamp
DDS::DataWriter, 42
register_type
DDS::DomainParticipant, 52
related_id
DDS::rpc::example::Foo_Reply, 239
release
DDS::LoanedSamples, 121
ReliabilityQosPolicy
DDS::ReliabilityQosPolicy, 96
ReliabilityQosPolicyKind
DDS, 320
Replier
DDS::rpc::Replier, 209
reply_topic_name
DDS::rpc::RequesterParams, 224
request_id
DDS::rpc::example::Foo_Request, 240
request_topic_name
DDS::rpc::RequesterParams, 224
RequestedDeadlineMissedStatus, 116
RequestedIncompatibleQosStatus, 116
Requester
DDS::rpc::Requester, 216
requester_listener
DDS::rpc::RequesterParams, 224
reserve
DDS::sequence, 148
resize
DDS::sequence, 148
ResourceLimitsQosPolicy
DDS::ResourceLimitsQosPolicy, 97
result
DDS::rpc::example::Foo_op1_Result, 236
DDS::rpc::example::Foo_op2_Result, 238
resume
DDS::rpc::ServiceEndpoint, 226
resume_publications
DDS::Publisher, 60
return_
DDS::rpc::example::Foo_op1_Out, 236
DDS::rpc::example::Foo_op2_Out, 238
return_datareader_qos
DDS Quality of Service, 6
return_datawriter_qos
DDS Quality of Service, 6
return_loan
DDS::DataReader, 34
DDS::LoanedSamples, 121
return_loaned_value
DDS::DynamicData, 162

return_participant_qos
DDS Quality of Service, 6
return_participantfactory_qos
DDS Quality of Service, 6
return_publisher_qos
DDS Quality of Service, 6
return_subscriber_qos
DDS Quality of Service, 6
return_topic_qos
DDS Quality of Service, 7
ReturnCode_t
DDS, 318
RpcQosPolicy
DDS::RpcQosPolicy, 98
rx_init_buffer_size
CoreDX_UdpTransportConfig, 202
rx_max_buffer_size
CoreDX_UdpTransportConfig, 202
rx_meta_multicast
CoreDX_UdpTransportConfig, 202
rx_user_multicast
CoreDX_UdpTransportConfig, 202

SHARED_OWNERSHIP_QOS
DDS, 320
sample_count
DDS_CacheStatistics, 142
sample_read_count
DDS_CacheStatistics, 142
sample_state
DDS_SampleInfo, 145
SampleIdentity_t
DDS::SampleIdentity_t, 133
SampleInfo, 143
SampleLostStatus, 116
SampleRejectedStatus, 117
sec
DDS_Time_t, 135
SequenceNumber_t
DDS::SequenceNumber_t, 134
service_name
DDS::rpc::RequesterParams, 224
ServiceStatus
CoreDX DDS RPC, 17
set_boolean
CDX::BooleanDynamicType, 248
set_boolean_value
DDS::DynamicData, 162
set_boolean_values
DDS::DynamicData, 162
set_byte_value
DDS::DynamicData, 163
set_byte_values
DDS::DynamicData, 163
set_char
CDX::CharDynamicType, 249
set_char32_value
DDS::DynamicData, 163
set_char32_values
DDS::DynamicData, 163
set_char8_value
DDS::DynamicData, 163
set_char8_values
DDS::DynamicData, 164
set_complex_value
DDS::DynamicData, 164
set_constant
CDX::EnumDynamicType, 254
set_default_datareader_qos
DDS::Subscriber, 69
set_default_datawriter_qos
DDS::Publisher, 61
set_default_field
CDX::UnionDynamicType, 271
set_default_participant_qos
DDS::DomainParticipantFactory, 55
set_default_publisher_qos
DDS::DomainParticipant, 53
set_default_subscriber_qos
DDS::DomainParticipant, 53
set_default_topic_qos
DDS::DomainParticipant, 53
set_discriminator
CDX::UnionDynamicType, 271
set_double
CDX::DoubleDynamicType, 250
set_element
CDX::ArrayDynamicType, 246
CDX::SequenceDynamicType, 260
set_element_type
CDX::ArrayDynamicType, 247
CDX::SequenceDynamicType, 260
set_enabled_statuses
DDS::StatusCondition, 65
set_expression_parameters
DDS::ContentFilteredTopic, 25
set_field
CDX::StructDynamicType, 265
CDX::UnionDynamicType, 272
set_field_label
CDX::UnionDynamicType, 272
set_field_num_labels
CDX::UnionDynamicType, 272
set_float
CDX::FloatDynamicType, 255
set_float32_value
DDS::DynamicData, 164
set_float32_values

DDS::DynamicData, 164
set_float64_value
DDS::DynamicData, 164
set_float64_values
DDS::DynamicData, 165
set_int16_value
DDS::DynamicData, 165
set_int16_values
DDS::DynamicData, 165
set_int32_value
DDS::DynamicData, 165
set_int32_values
DDS::DynamicData, 165
set_int64_value
DDS::DynamicData, 165
set_int64_values
DDS::DynamicData, 166
set_length
CDX::ArrayDynamicType, 247
CDX::SequenceDynamicType, 260
set_listener
DDS::DataReader, 34
DDS::DataWriter, 42
DDS::DomainParticipant, 53
DDS::Publisher, 61
DDS::Subscriber, 69
DDS::Topic, 71
set_long
CDX::LongDynamicType, 256
set_longlong
CDX::LongLongDynamicType, 257
set_max_length
CDX::ArrayDynamicType, 247
CDX::SequenceDynamicType, 261
CDX::StringDynamicType, 263
set_num_constants
CDX::EnumDynamicType, 254
set_num_fields
CDX::StructDynamicType, 266
CDX::UnionDynamicType, 273
set_octet
CDX::OctetDynamicType, 258
set_qos
DDS::DataReader, 35
DDS::DataWriter, 42
DDS::DomainParticipant, 53
DDS::DomainParticipantFactory, 55
DDS::Publisher, 61
DDS::Subscriber, 69
DDS::Topic, 72
set_query_parameters
DDS::QueryCondition, 62
set_short
CDX::ShortDynamicType, 262
set_string
CDX::StringDynamicType, 264
set_string_value
DDS::DynamicData, 166
set_string_values
DDS::DynamicData, 166
set_trigger_value
DDS::GuardCondition, 57
set_uint16_value
DDS::DynamicData, 166
set_uint16_values
DDS::DynamicData, 166
set_uint32_value
DDS::DynamicData, 167
set_uint32_values
DDS::DynamicData, 167
set_uint64_value
DDS::DynamicData, 167
set_uint64_values
DDS::DynamicData, 167
set_ulong
CDX::ULongDynamicType, 267
set_ulonglong
CDX::ULongLongDynamicType, 268
set_ushort
CDX::UShortDynamicType, 274
set_value
CDX::EnumDynamicType, 254
DDS::AnnotationDescriptor, 150
set_wstring_value
DDS::DynamicData, 167
set_wstring_values
DDS::DynamicData, 168
simple_requester_listener
DDS::rpc::RequesterParams, 224
so_rcvbuf
CoreDX_LmtTransportConfig, 197
CoreDX_UdpTransportConfig, 202
so_sndbuf
CoreDX_LmtTransportConfig, 197
CoreDX_UdpTransportConfig, 202
state_flags
DDS_CacheStatistics, 142
StatusMask
DDS, 318
strict_match
PeerParticipantQosPolicy, 94
subscriber
DDS::rpc::RequesterParams, 224
SubscriptionMatchedStatus, 117
suspend_publications
DDS::Publisher, 61
swap
DDS::sequence, 148

TOPIC_PRESENTATION_QOS
DDS, 320
TRANSIENT_DURABILITY_QOS
DDS, 319
TRANSIENT_LOCAL_DURABILITY_QOS
DDS, 319
TYPECODE_ALIAS
dds_dtype.hh, 290
TYPECODE_ARRAY
dds_dtype.hh, 290
TYPECODE_BOOLEAN
dds_dtype.hh, 290
TYPECODE_CHAR
dds_dtype.hh, 290
TYPECODE_DOUBLE
dds_dtype.hh, 290
TYPECODE_ENUM
dds_dtype.hh, 290
TYPECODE_FLOAT
dds_dtype.hh, 290
TYPECODE_LONGDOUBLE
dds_dtype.hh, 290
TYPECODE_LONGLONG
dds_dtype.hh, 290
TYPECODE_LONG
dds_dtype.hh, 290
TYPECODE_OCTET
dds_dtype.hh, 290
TYPECODE_SEQUENCE
dds_dtype.hh, 290
TYPECODE_SHORT
dds_dtype.hh, 290
TYPECODE_STRING
dds_dtype.hh, 290
TYPECODE_STRUCT
dds_dtype.hh, 290
TYPECODE_ULONGLONG
dds_dtype.hh, 290
TYPECODE_ULONG
dds_dtype.hh, 290
TYPECODE_UNION
dds_dtype.hh, 290
TYPECODE_USHORT
dds_dtype.hh, 290
TYPECODE_VALUE_PARAM
dds_dtype.hh, 290
TYPECODE_VALUE
dds_dtype.hh, 290
TYPECODE_WCHAR
dds_dtype.hh, 290
TYPECODE_WSTRING
dds_dtype.hh, 290
take
DDS::DataReader, 35
take_instance
DDS::DataReader, 36
take_next_instance
DDS::DataReader, 36
take_next_instance_w_condition
DDS::DataReader, 36
take_next_sample
DDS::DataReader, 36
take_replies
DDS::rpc::Requester, 219
take_reply
DDS::rpc::Requester, 219, 220
take_request
DDS::rpc::Replier, 210
take_requests
DDS::rpc::Replier, 211
take_w_condition
DDS::DataReader, 37
ThreadModelQosPolicy, 99
TimeBasedFilterQosPolicy
DDS::TimeBasedFilterQosPolicy, 100
to_shared
DDS::LoanedSamples, 122
TopicDataQosPolicy
DDS::TopicDataQosPolicy, 101
TransportPriorityQosPolicy
DDS::TransportPriorityQosPolicy, 101
tx_max_packet_size
CoreDX_TcpTransportConfig, 199
CoreDX_UdpTransportConfig, 202
tx_meta_multicast
CoreDX_UdpTransportConfig, 202
tx_meta_unicast
CoreDX_UdpTransportConfig, 202
type
DDS::AnnotationDescriptor, 151
TypeCodeKind
dds_dtype.hh, 290
TypeConsistencyEnforcementQosPolicy
DDS::TypeConsistencyEnforcementQosPolicy, 102
TypeConsistencyKind
DDS, 320
TypeDescriptor
DDS::TypeDescriptor, 192
TypecodeQosPolicy
DDS::TypecodeQosPolicy, 102

unbind
DDS::rpc::ServiceProxy, 229
unknownEx
DDS::rpc::example::Foo_op1_Result, 236
DDS::rpc::example::Foo_op2_Result, 238
unknownOp
DDS::rpc::example::Foo_Return, 241

unregister_instance
DDS::DataWriter, 42
unregister_instance_w_timestamp
DDS::DataWriter, 42
use_ipv4
CoreDX_UdpTransportConfig, 203
use_ipv6
CoreDX_UdpTransportConfig, 203
user_multicast_address_v4
CoreDX_UdpTransportConfig, 203
user_multicast_address_v6
CoreDX_UdpTransportConfig, 203
UserDataQosPolicy
DDS::UserDataQosPolicy, 103

VOLATILE_DURABILITY_QOS
DDS, 319
valid_data
DDS_SampleInfo, 145
value
CDX::EnumConstant, 253
DDS::BuiltinTopicKey_t, 131
DDS::GUID_t, 132
value_type
DDS::LoanedSamples, 121
DDS::SampleIterator, 125
DDS::SharedSamples, 128
view_state
DDS_SampleInfo, 145

wait
DDS::WaitSet, 74
wait_for_acknowledgments
DDS::DataWriter, 42
DDS::Publisher, 61
wait_for_replies
DDS::rpc::Requester, 220, 221
wait_for_requests
DDS::rpc::Replier, 211
wait_for_service
DDS::rpc::ServiceProxy, 229, 230
wait_for_services
DDS::rpc::ServiceProxy, 230
write
DDS::DataWriter, 43
write_w_timestamp
DDS::DataWriter, 43
WriterDataLifecycleQosPolicy, 103

X-Types Dynamic Types, 19